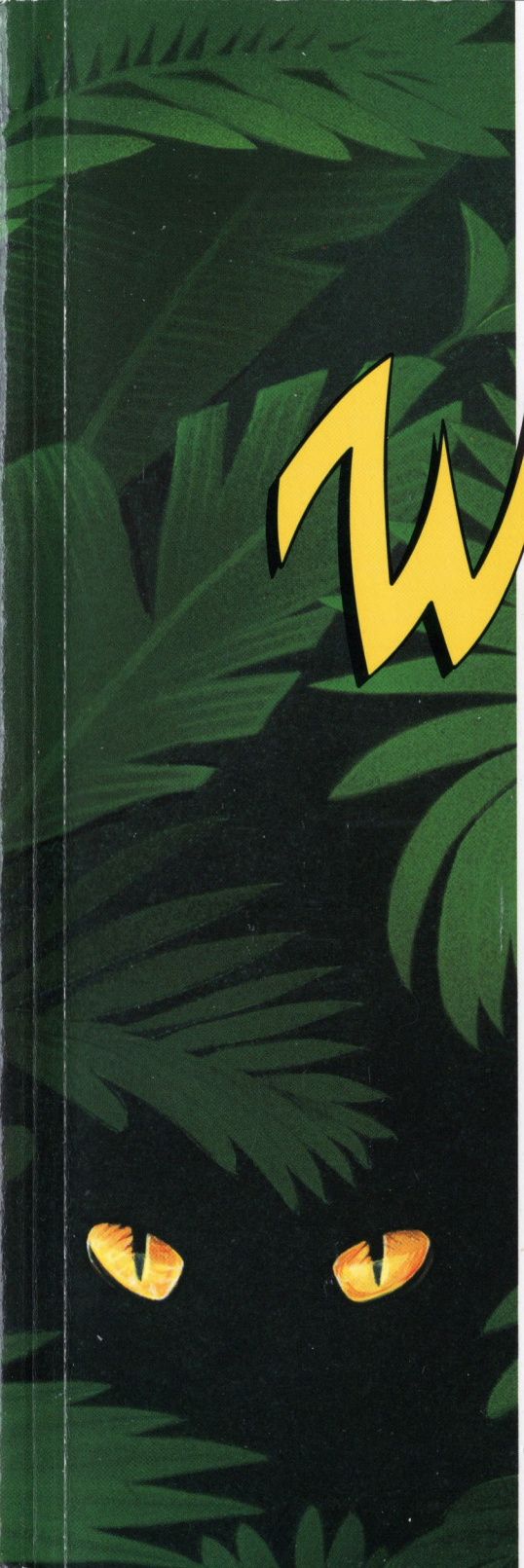


Wildcat! 4

wcCODE Sysop Guide

Keep this portion as your copy and Proof of Ownership

Personal Registration Number

Customer Registration Card

wcCODE Registration Card

First Name: _____ Last Name: _____

Company _____ Ph: (____) _____

Address: _____

_____

City: _____ State: _____ Zip: _____

Outside USA, Country*: _____ Postal Code: _____

Date Purchased: _____ Ver. #: _____

Where Purchased: _____

Personal Registration Number

* If you are mailing this card from outside the U.S., please send it in an envelope with proper postage.

Complete the attached card and drop it in the mail today! With most products, registering ownership is not essential. With BBS software, it's important to know when new versions are available. We can notify you only if we have your name and address.

By registering your copy of *wcCODE*, you'll receive:

FREE! Enrollment in our technical support program.

FREE! Access to the *MSI HQ* Registered Support BBS to obtain value-added utilities.

FREE! Notification of major program updates.

There's no reason not to register now! The postage is pre-paid.

Note to customers upgrading from a prior release: You need to return this card to continue to receive update information!

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 10468 BAKERSFIELD CA

POSTAGE WILL BE PAID BY THE ADDRESSEE

MUSTANG SOFTWARE INC
PO BOX 2264
BAKERSFIELD CA 93303-9943

wcCODE

Wildcat! Custom Online Development Engine

(07/01/94 printing)

P.O. Box 2264

Bakersfield, CA 93303

Tech. Support (805) 873-2550 — Office & Sales (805) 873-2500

Fax (805) 873-2599 — BBS (805) 873-2400

Orders only (800) 999-9619

The information in this document is subject to change without notice and should not be construed as a commitment on the part of Mustang Software, Inc.

Wildcat!, MSI and the MSI Horse Logo are trademarks of Mustang Software, Inc. Other products and corporate names may be trademarks or registered trademarks of other companies, and are used in this document for illustrative purposes only, and to the owners benefit, without an intent to infringe. The names of companies, persons and products used in this documentation are fictitious unless otherwise noted, and are included solely to document the program.

Copyright © 1986, 1994 Mustang Software, Inc. All rights reserved. No part of this document may be reproduced, transmitted or transcribed in any form or by any means, electronic or mechanical, including photocopying or recording, without the written permission of Mustang Software, Inc.

Table of Contents

wcCODE 1
Software License Agreement 5
Limited Warranty 8
Copyright 9
Satisfaction Guarantee 9
Installation 10
Customer Access 15
Satisfaction Guarantee 19
About This Guide 20

1 - The *wcCODE* IDE 23
Before we start, a word or two of caution 25
Life at the cutting edge 26
The windows 27
The *wcIDE* User Interface 29
Dialog Boxes 30
Using a Mouse with the IDE 31
Using the Keyboard with the IDE 32
The menus 33

2 - Programming 41
The basic structure of *wcCODE* 43
General Program Information 44
When the over-view is over 51
Preamble to programming 52
We bid a fond farewell 78

3 - Commands 79
wcCODE Command Reference 81

Appendix 427
Reserved Keywords for *wcCODE* 429
@codes 440
Color Codes 445

ANSI codes 446
Compile-time errors 448
Run-time errors 456
Editor commands used in the IDE 457
RIP commands 459
The date and time compressed formats used by *wcCODE* 462
Boolean bitwise operations 465
Internal data structures 467

Index 479

Software License Agreement

Important: Please read this page before using the disks. By opening the package containing the disks you acknowledge your agreement to the terms of this license.

Refund

If you do not wish to follow the terms of this license agreement, you may obtain a full refund by returning this package to your authorized dealer within three (3) days, provided you have not opened the sealed disk envelope.

Definitions

The term "Software" as used in this agreement means the full system and all utility computer programs contained on the disks in this package, together with any updates subsequently supplied by Mustang Software.

The term "Software Copies" means the actual copies of all or any portion of the Software, including back-ups, updates, merged or partial copies permitted here under or subsequently supplied by Mustang Software.

The term "Related Materials" means all the printed materials provided in this package or later supplied by Mustang Software for use with the Software.

Permitted uses

You may:

Use the Software on a single computer or on multiple computers that are connected by direct cable to the same network server.

Install the Software onto a permanent storage device (a hard disk drive).

Make non-operational archival copies of the Software for backup purposes.

Make one working copy of the Software Program. If the working copy will no longer work properly, the Licensee may make another working copy of

the Software Program but only if the inoperable working copy is destroyed. In addition, all the information appearing on the original disk labels (including the copyright notice) must be copied onto the working copy labels.

This license gives you certain limited rights to use the Software. You do not become an owner of and Mustang Software retains title to all the Software, Software Copies and Related Materials. In addition, you agree to use reasonable efforts to protect the Software from unauthorized use, reproduction, distribution, or publication. All rights not specifically granted in this license are reserved by Mustang Software.

Uses not permitted

You may not:

Make copies of the Software, except as permitted above.

Make use of the Software or any part of the Software on more than one computer at a time.

Make copies of the Related Materials.

In accordance with the Computer Software Rental Act of 1990, rent, lease, sub-license, lend or transfer the Software, Software Copies, Related Materials or your rights under this license without the prior written consent of Mustang Software, Inc.

Alter, decompile, disassemble, or reverse engineer the Software.

Remove or obscure the Mustang Software copyright and trademark notices.

This software and its documentation may not be provided by a "backup service" or any other vendor which does not provide an original package as composed by Mustang Software, Inc. including but not limited to all original documentation, insertions, registration cards, and software.

Duration

This agreement is effective from the day you open the sealed disk package. Your license continues for twenty-five years or until you return to Mustang Software the original disks and any back-up copies, whichever comes first.

If you breach this agreement, we can terminate this license by notifying you in writing. You will be required to return all Software, Software Copies and Related Materials. We may also enforce our other legal rights.

General

This agreement represents our entire understanding and agreement regarding the Software, Software Copies and Related Materials and supersedes any prior purchase order, communications, advertising, or representations.

This license may only be modified in a written amendment signed by an authorized Mustang Software officer. If any provision of this agreement shall be unlawful, void, or for any reason unenforceable it shall be deemed severable from, and shall in no way affect the validity or enforceability of the remaining provisions of this agreement. This agreement shall be governed by California law. You acknowledge that you have read every provision of this contract. By opening the envelope containing the disks, you acknowledge that you have read and agree to this contract.

Limited Warranty

MUSTANG SOFTWARE WARRANTS TO THE ORIGINAL LICENSEE OF A REGISTERED PRODUCT THAT THE PROGRAM DISK(S) ON WHICH THE PROGRAM IS RECORDED WILL BE FREE FROM DEFECTS IN MATERIALS AND WORKMANSHIP UNDER NORMAL USE AND SERVICE FOR A PERIOD OF NINETY (90) DAYS FROM THE DATE OF DELIVERY AS EVIDENCED BY A COPY OF YOUR RECEIPT. MUSTANG SOFTWARE'S ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE REPLACEMENT OF THE DISK NOT MEETING MUSTANG SOFTWARE'S LIMITED WARRANTY IF RETURNED TO MUSTANG SOFTWARE DURING THE 90 DAY PERIOD.

EXCEPT FOR THE FOREGOING, THIS PRODUCT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PROGRAM IS ASSUMED BY YOU. MUSTANG SOFTWARE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, RESULTS OF, MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE OF THE PRODUCT. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT MUSTANG SOFTWARE OR ITS DEALERS) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. FURTHER, MUSTANG SOFTWARE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, OR THE RESULTS OF THE USE OF THE PROGRAM IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE; AND YOU RELY ON THE PROGRAM AND RESULTS SOLELY AT YOUR OWN RISK. MUSTANG SOFTWARE WILL NOT BE LIABLE FOR ANY DAMAGES, INCLUDING BUT NOT LIMITED TO SYSTEM DAMAGE, SERVICE, REPAIR, CORRECTION, LOSS OF PROFIT, LOST SAVINGS, OR ANY OTHER INCIDENTAL, CONSEQUENTIAL, OR SPECIAL DAMAGES OF ANY NATURE WHATSOEVER RESULTING FROM THE USE OR INABILITY TO USE THIS PRODUCT.

Copyright

The program *wcCODE* is NOT in the public domain. It is copyrighted by Mustang Software, Inc. All rights reserved. Copying, duplicating, selling or otherwise distributing this product is a violation of the law.

This manual is also copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent in writing from Mustang Software. Mustang Software, Inc. may make improvements and/or changes to both this manual and the program at any time.

Throughout this manual any commercial trade names mentioned are either trademarks or registered trademarks of their developers. Use of these trade names does not indicate an endorsement by the trademark owners.

Satisfaction Guarantee

Mustang Software wants you to be satisfied with your purchase of *wcCODE*. If for any reason you are not satisfied and wish to return *wcCODE*, you may do so within the first 30 days after purchase. Simply return the entire contents of the package, including the manual, disks, and registration card, to your dealer. Proof of purchase date is required. Returns direct to MSI can only be arranged if the product was purchased direct from MSI. Direct returns must be pre-approved and a return authorization number must be obtained in advance.

Installation

wcCODE comes with an installation program that automatically creates the necessary directories, and copies the program and support files you select to the correct location. You cannot install *wcCODE* simply by copying the files from the installation diskettes to your hard drive since the files are stored in compressed format.

You can use the installation program for an initial installation, or to re-install your *wcCODE* software. Before you begin installing *wcCODE*, we strongly recommend that you make a backup copy of the installation diskettes and keep them in a safe place. Review your DOS manual for help if you are unsure how to make backup copies of your diskettes.

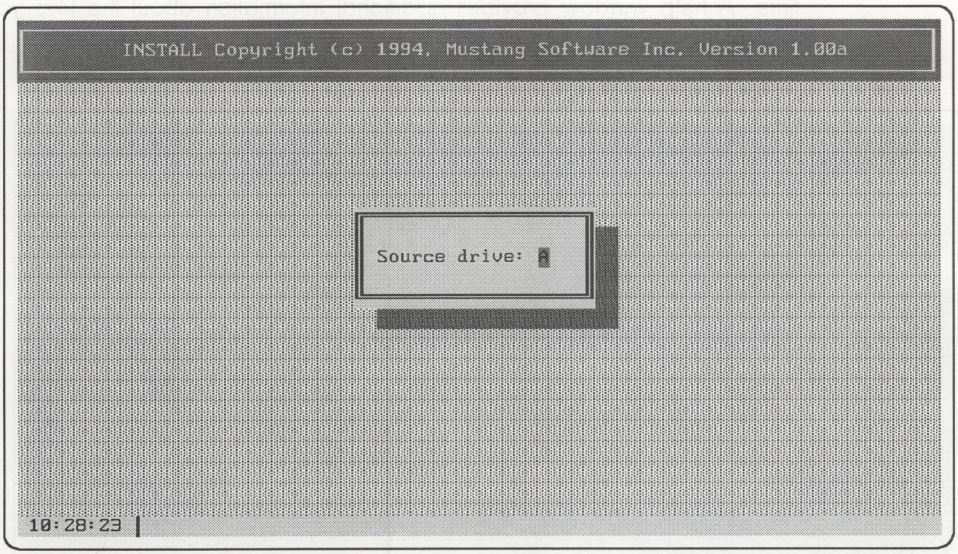
To install *wcCODE*, place the disk labeled "INSTALL" in your floppy drive and type

```
A: ENTER
```

If you are installing from a different drive, enter the proper drive letter. Then type

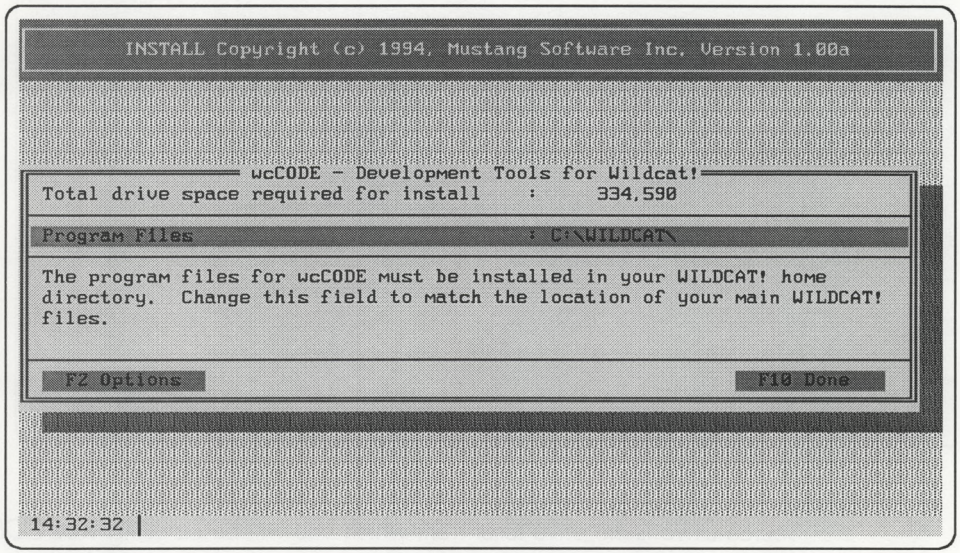
```
WCCINST ENTER
```

Your screen should now look something like this:



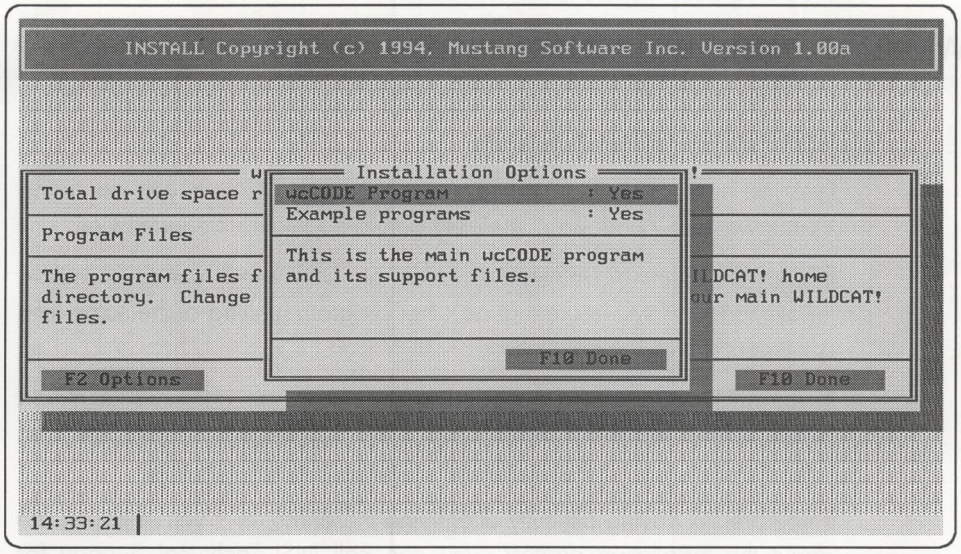
Source Drive is the drive letter from which you will be installing *wcCODE*. If you are installing from diskettes in floppy drive A:, enter A in this box. If you are installing from drive B:, enter B, and so on. Press ENTER to continue.

The next screen will show you the amount of drive space required for the installation, and allows you to select which files you want to install at this time. A help window provides additional information about the files you are selecting.



The correct place to install all *wcCODE* Program files is your *Wildcat!* home directory. The default directory is C:\WILDCAT\. If you have installed *Wildcat!* on a different drive or subdirectory, enter the correct information here.

The default installation installs *all* program and support files, and this is the recommended procedure for a first-time installation. Press F2 to see installation options for each set of files. As with the previous window, the Installation Options screen explains each option. Use your ↑ and ↓ keys to move the selection bar up and down the list, and press ENTER to change each option. Notice how the help information changes as you move the selection bar.



When you have finished selecting installation options, press F10 to return to the main installation screen. Press F10 again to begin the installation.

The installation process will copy the program files to your *Wildcat!* home directory, and will create any required subdirectories for configuration and support files. The paths created are the proper location for configuration and support files, and shouldn't be altered.

The installation program may ask you to insert additional diskettes to complete the installation process. Each original diskette is clearly labeled.

The installation program will display additional screens of tips and information while the *wcCODE* files are being copied to your hard drive.

In some cases, changes may have been made to the program after the manuals were printed. The final screen displayed by the installation program shows you important information on last-minute changes to the program or documentation. You have the option to print this file right now. We recommend you do so, and keep the printed copy with your *wcCODE* User Guide for future reference.

What to do if you have problems installing *wcCODE*

If you have problems installing *wcCODE*, here are a few things to check before calling Technical Support.

First, be sure you are installing *wcCODE* from DOS. If you are using Windows, exit Windows completely before beginning the installation.

If you are running DESQview, close all DESQview windows before installation. This will insure that no other program running at the same time is interfering with the installation.

If you think you may have bad diskettes, try to use both sets of media before calling Technical Support. *wcCODE* is provided on both 5¼" and 3½" diskettes. It is rare for both sets of diskettes to be defective.

If you are still having difficulty after trying these tips, or if you're just unsure about the installation process, you can call MSI Technical Support for assistance.

Customer Access

Mustang Software, Inc. strongly believes in providing quality software at a reasonable price. We back our products with technical help when our registered users need it. Please send in your registration card to enroll in our technical support plan.

Until we receive your registration information, our staff cannot provide you with technical support via our BBS or by phone.

Benefits of sending your registration

- Free technical support by phone Monday through Friday between 9 a.m. and 5 p.m. Pacific time
- Automatic notification of major program updates
- Additional program updates at a reduced rate
- 24-hour access to the private multi-line Registered User's HQ BBS at Mustang Software
- BBS access to download utility programs written by MSI staff and other authors which enhance and add features

Technical Support Services

wcCODE is designed to be easy to use, and this manual should contain the answers to most of your questions. Read it first. Make use of the F1 help key for tips and explanations. If you're still stumped, technical support is only a phone call away.

MSI HQ BBS

You may first want to call our private *Wildcat!* BBS for Registered Users at 805-873-2400, and leave a quick question for our technical staff. This is a particularly useful way to receive expert guidance with the more advanced features.

Note: You will not have full access on your first call since we need you to fill out the user database information such as your password, etc. The

System Operator (Sysop) will raise your security access as soon as possible, usually within 24-48 hours, if your registration information is on file.

On-line services

CompuServe

Another alternative is CompuServe where we are a part of the PC Vendor Support Forum. You reach us by typing GO PCVENA and then selecting SubTopic 9.

GEnie

You can also find us on GEnie, the on-line service from General Electric. Go to the MUSTANG RoundTable with the command MOVE 680.

America Online

MSI's support section on America Online is called MUSTANG and can be reached with the keyword MUSTANG.

Internet

For technical support via Internet, write to support@mustang.com.

Voice support

If you are still unable to find the answer to a question—or if you need a quick explanation, please call us Monday to Friday between 9 a.m. and 5 p.m. Pacific time at 805-873-2550.

When calling for support please:

- Have your registration number handy (it is printed inside the cover of your *Wildcat!* manual).
- Record your Registration number here:

_____

- Be at your computer with your manual handy.
- Be ready to describe the problem in detail.
- If possible, be ready to duplicate the problem on your system.
- You can call technical support directly at (805) 873-2550.

Support is not available by mail or Fax.

MSI SupportNET

We also have a large number of BBS systems carrying the MSI SupportNET echo conferences throughout the US and Canada, and around the world. These BBSs can be found in the United Kingdom, Holland, Australia, France, and Norway. The MSI SupportNET contact points outside the U.S. are:

Location	Distributor	BBS Phone
Australia	Banksia Info. Tech.	+61 2-418-7693
France	Maxotex	+33 1-45-34-6430
Scandinavia	PC Security	+47 2-58-3358
United Kingdom	Telesystems	+44 49-489-1903

Questions Other Than Technical Support

If you have questions related to shipping or other areas handled by our office staff you should contact us at 805-873-2500. Our office personnel can help with changes to your registration information and handle all non-support questions regarding our product line.

We also have a toll-free number for questions regarding new product purchases or updates. Just call 1-800-999-9619. Our mailing address is Mustang Software, Inc., P.O. Box 2264, Bakersfield, CA 93303.

Software Updates

wcCODE is constantly undergoing enhancements and revisions. This is normal software maintenance. We encourage you to keep your software updated—we can provide the best support to users who are running the current major release. The cost is minimal and the benefits are great.

Satisfaction Guarantee

Mustang Software wants you to be satisfied with your purchase of *Wildcat! BBS*.. If for any reason you are not satisfied and wish to return *Wildcat! BBS*, you may do so within the first 30 days after purchase. Simply return the entire contents of the package, including the manual, disks, and registration card, to your dealer. Proof of purchase date is required. Returns direct to MSI can only be arranged if the product was purchased direct from MSI. Direct returns must be pre-approved and a return authorization number must be obtained in advance.

About This Guide

This manual describes *wcCODE* operation and features at the time of publication, and does not reflect changes made to the program after the manual was completed. Be sure to check the README file on your *wcCODE* installation disks. This file contains important information about last-minute changes in the program, additional features, and errors or omissions in the manual.

This *wcCODE* program reference guide is designed to help you write programs that will perform just like native instructions within *Wildcat!*. This guide is not designed or intended to teach you programming or the BASIC programming language. However the following introductory chapter should give you more than enough basic understanding of *wcCODE* to get you well on your way. Even if you have never programmed before or even thought you could, we believe that after a little practice you will very comfortable with this programming language.

Chapter One is an overview of the *wcCODE* Integrated Development Environment (IDE) we have provided as a workbench-editor. The IDE comes with an integrated compiler for smoother program development.

Chapter Two provides an overview of *wcCODE* and an introduction to the programming approach needed to make you a successful *wcCODE* developer. You will find *wcCODE* to be a very powerful and flexible language that can completely customize your *Wildcat!* BBS any way you would like it to be. With this programming language at your command, you will be limited only by your imagination.

Chapter Three contains a complete list of *wcCODE* commands and is designed to be used as both a reference and a minimal tutorial on the command language structure. You will also find lots of useful code-fragments that you can copy for use in your own programs. Following the commands in this chapter we outline several sample programs we have included with the *wcCODE* package.

Getting additional information

In addition to the information in this guide there are numerous sources for help and training on the BASIC programming language, the underlying language of *wcCODE*. Your favorite bookstore or computer user group can be an invaluable source of help. Look for a tutor on the BASIC implementation used by Quick Basic from Microsoft and you will be heading in the right direction. You may find however that there is sufficient information provided in the next few chapters so that a little practice with *wcCODE* might be all that you need.

1 - The *wcCODE* IDE

He who is only wise lives a sad life.

Voltaire

In this chapter

Before we start, a word or two of caution 25
Life at the cutting edge 26
The windows 27
The *wcIDE* User Interface 29
Dialog Boxes 30
Using a Mouse with the IDE 31
Using the Keyboard with the IDE 32
- Moving the cursor and selecting options 32
- Executing commands from the keyboard 32

The menus 33
- File 33
- Edit 34
- Search 36
- Compile 37
- Options 38
- Window 38
- Help 39
- Let's move along now. 40

Before we start, a word or two of caution

The *wcCODE* programming language can be a very powerful tool in the hands of a competent programmer. It must be your responsibility to ensure the commands you are writing with *wcCODE* will do exactly what is intended, rather than hoped.

Very serious damage to your *Wildcat!* system could result from careless, or poorly designed *wcCODE* programs. The suitability for the task and any results arising from the use of any *wcCODE* program must be entirely the responsibility of the user.

The language contains commands that could completely wipe clean an entire hard drive or array of drives. The *Wildcat!* databases are easily accessible via *wcCODE* commands and they too could suffer a similar fate, unless sufficient care is taken in the design and testing of all *wcCODE* programs.

Mustang Software Inc. cannot be held liable, or responsible for any damages or detrimental results or outcomes arising from the use of any *wcCODE* program regardless of who wrote the code.

Life at the cutting edge

To make editing your programs easier, Mustang has included in the *wcCODE* package, an Integrated Development Environment (IDE) program called WCIDE.EXE. This programmer's interface will allow editing and compiling of your *wcCODE* programs in a single working environment. The IDE has a fully functional mouse interface that you will find is very intuitive to use.

The use of the *wcIDE* is very straight forward, in fact almost self explanatory. The majority of keyboard commands for text editing, are based on the WordStar command set. You may or may not be familiar with this particular system but a large number of programming editors still support the WordStar command set. Similarly, an equally vast number of programmers continue to use the commands.

The *wcCODE* IDE is no exception in subscribing to this general acceptance of the WordStar command set. For a complete list of the supported WordStar commands refer to the appendix in the back of the manual.

In the following text, when describing a menu option, the associated hot-key will be shown towards the middle of the text, adjacent to the menu option. If there is a WordStar command for that option, it too, will be shown to the extreme right of the option name. For example the file menu when opened presents several options, one of which is...

Save	F2	CTRL K S
Menu option	Hot-key	WordStar command

The windows

Let's first look at the screen when you start up the IDE. The menu bar across the top of the screen displays all of the main menu options. You can access these menus by pressing the ALT key plus the highlighted menu character, or you can click on the menu name with the mouse. A third option for activating the main menu bar is to press the F10 key.

The bottom row of the screen is called the status line. It will display several hot-keys, some status indictors and the current time so you don't over-stay your time at the keyboard writing *wcCODE* programs. When a menu from the menu bar is open, then the status line will show a brief description of each option as you move the selection bar over them.

Status line indicators for the keys that can be toggled into different states. The keyboard indicator byte in DOS is read at startup to determine the condition of these toggles when the IDE is first launched.

Key	Indicator	Description
CAPS LOCK	CAPS	All upper case characters.
CAPS LOCK		Mixed case characters.
NUM LOCK	NUM	Display numbers.
NUM LOCK		Arrow keys move the cursor.
INS	INS	Insert text.
INS	OVR	Overwrite existing text.

Once a menu is open, to select an option use a quick double click of the mouse button, or simply press the highlighted character of the option to activate it. Some menu options will also have a 'hot key' indicated on the right-hand side of the panel. This 'hot-key' can be used at any time when you are editing program text. The hot-key is intended as a short-cut to opening the menu and selecting the option by name.

Other menu items will have three dots or periods following them to indicate there are additional information screens or dialog boxes to

follow. Some of these will contain conditional-options, where there will be one or several data fields or check boxes. For handling the data in these dialog boxes see the section titled, "dialog boxes" a little further on in the section.

The *wcIDE* User Interface

For ease of use and efficiency, the IDE is designed around a system of windows, menus and dialog boxes. Beginner users will probably be most comfortable navigating around the IDE by using the pull-down menus and dialog boxes. The more advanced programmers can make use of the keyboard shortcuts to go directly to the desired functions.

Many of the IDE functions can be activated from the keyboard, or by using a mouse or other pointing device. For each option that can be selected with a mouse, there is usually an equivalent keyboard shortcut.

Dialog Boxes

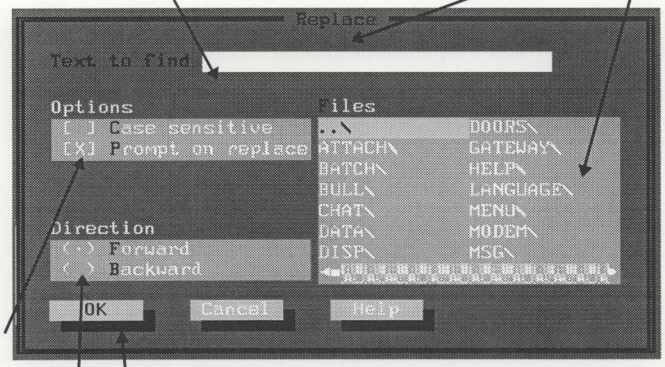
A dialog box contains several elements, appearing in various combinations depending on the selection. All dialog boxes can be closed by using the ENTER key. Likewise the operation can be canceled with the ESC key. The following diagram shows a dialog box with all possible elements displayed.

Text boxes are used to enter, save, and list text entries associated with the dialog box.

The **title bar** identifies the dialog box by function.

List boxes show you a list of items from which you select your choice. An example might be the window list, which allows you to choose from available active windows. A List box has scroll bars to display more items than the box can display on screen. Items in a list box can be selected by the mouse or the keyboard.

Check boxes are used to list and select multiple items from a list. Options can be toggled on or off by moving the highlight bar over the desired item and pressing SPACE or clicking the left mouse button.



Option boxes are used to list and select one item from a list of available options. The option can be selected by moving the 'dot' between the parentheses by using the ↑ ↓ keys.

Control buttons cause the immediate selection of the function assigned to the button. This may lead to another dialog box, or it may execute the function directly.
In some cases, a control button will ask you to answer "Yes" or "No" to confirm your selection of an IDE function.

Using a Mouse with the IDE

Anyone familiar with the standard mouse interface found in many popular DOS and Windows applications will find the IDE mouse interface easy to use and you may skip the following section on mouse usage.

The mouse pointer will be visible on most screens as a large block character. By moving the mouse around on your desktop, you can observe the movement of the mouse cursor on the screen.

The general rule for item selection with a mouse is that clicking the left button selects the item that is under the mouse cursor. The selection may cause another menu to appear, from which you can make further selections. As an alternative you may see a dialog box in which you can select further items or enter text from the keyboard to fill in a data entry field.

Using the Keyboard with the IDE

Most mouse movements and selections can also be made from the keyboard. Cursor movements and menu item selections generally follow the conventions established by programs such as Windows.

Moving the cursor and selecting options

Your arrow keys and tab keys are used to move around the various menu items and dialog boxes. For example, your arrow keys move a highlight bar right and left along the menu bar. Press [ENTER] to select the sub-menu for each item under the highlight bar.

In dialog boxes, the [⇥] (Tab) and [SHIFT] [⇥] keys move forwards and backwards between the various data entry boxes and radio buttons. The arrow keys are used to highlight and select individual items within a data entry box.

Executing commands from the keyboard

Most of The IDE's menu commands can be executed with a combination of "Alt" or "Ctrl" keys. To display the window-list, for instance, hold down your [ALT] key and [0] (zero) key at the same time. The window-list window will then pop up on the screen. From there, you can select a window to make active by scrolling up and down through the list, or you may select another option by pressing the key highlighted in an option button. You may also use your [⇥] key to move from option to option, then press [ENTER] to execute your choice.

As you become familiar with the IDE, you'll probably find yourself using these keyboard shortcuts to go directly to the function you wish to use.

The menus

File

New

Used for starting out a new document. It gives you a clean editing window ready for your masterpiece.

Open... F3

This option will present a dialog box with the list of all available files with the .WCC extension. This is the default file extension for *wcCODE* source code files. You make a selection by double clicking the mouse on a file name or enter the file name in the entry box.

If you are starting a new document then type in the name you want to call it and you are on your way to another piece of programming perfection.

Save F2 CTRL K S

Save the current file under the same name as it was loaded. If it is a new file created by entering the name in the entry box during the "Open" menu option, then "Save" will simply update the stored version on disk.

If the file was created using the "New" menu option then it will prompt you for a name. If the name already exists then another box will open and ask you whether to over-write the existing file.

Save as... CTRL K N

Save your work under a new name that you enter into the entry box. This option is useful if you are developing a new version of an old program and you don't want to modify the existing working copy. You would load the old copy then immediately "Save as" a new name and then continue to work on it.

Save all

Save all currently open files. Since the IDE can support dozens of open files at a time this option is a quick method of ensuring that changes are stored to disk. This saves you having to select each window and press the F2 key for a Save.

Print

Print will, as the name aptly describes, print the entire current document. This will send all output to the PRN device which must be mapped correctly if you are running on a network.

DOS shell

This option allows you to drop back to DOS to perform DOS commands. You will need to type "Exit" to return to the IDE when finished with the DOS shell.

Exit

ALT X

Close all windows and exit the IDE. If you have made changes to your document then you will be asked if you want to save first. The IDE will save the names of the active files in its own configuration file. When you re-start the IDE it will reload these files and present you with your work area the way it was the last time you used it.

Highlighted numbers

The file menu maintains a list of the last five files opened in the IDE. You can short-cut the "open" process by selecting the highlighted number on the left of this list.

Edit

The Edit menu is used in conjunction with a marked block of text. To mark the text you can use the F7 key to indicate the current cursor position as the start of the marked block. Move the cursor to the end of the section and press the F8 key to mark the end of the block. The selected text will

then be highlighted. To un-highlight the text press F7 and then F8 without moving the cursor to mark a block with an effective length of zero.

Alternately you could use the WordStar commands for marking a block, CTRL K B to mark the beginning, CTRL K K to mark the end, with CTRL K H used to hide and un-hide the marked block. Once the block of text has been marked, the cut and paste options will cut or copy it to the clipboard. The SHIFT key used in conjunction with the ↑ ↓ ← → keys will also mark a block of text.

The clipboard is a temporary area that will contain only the last cut or copy operation. The information on the clipboard can be pasted as many times as you like but each subsequent cut or copy will replace the previous one.

To move a marked block of text you will need to cut it, then move the cursor to the new location and paste it. An alternative to this is to mark the block, move the cursor to the new location and use the WordStar command CTRL K V. This will immediately move the block of text to the new location but it will not place a copy on the clipboard.

While the WordStar option is a faster operation than cut and paste, it works only within the current document, whereas cut and paste can be used across windows. You can cut, or copy a marked block in one window, swap to another window and paste into it.

Cut

SHIFT DEL

Remove the marked section of text and store it on the clipboard.

Copy

CTRL INS

Make a copy of the marked text on the clipboard.

Paste

SHIFT INS

Insert the text from the clipboard into the current document at the cursor location.

Clear

CTRL DEL CTRL K Y

Remove the marked text. It is completely discarded and not copied to the clipboard but the clipboard will still contain any text that was previously copied to it.

Search

Find...

CTRL Q F

Find specific information by opening a dialog box and presenting you with an entry box to fill in with the required text. The options within the dialog box will affect the way the "Find" is carried out.

Options

Sub-option	Options	Action to take
Options	Case sensitive	Match the case of the text exactly as you typed it in the entry box.
Direction	Forward	Search from the cursor towards the end of the document.
	Backward	Search from the cursor towards the beginning of the document.
Range	From the cursor	The search will start exactly where ever the cursor was when you opened the "Search" menu. Depending on the "Direction", the search will terminate at the beginning or the end of the file.
	Global	The search will include the entire file.
	Selected text	The search in the marked block of text only.

Replace...

CTRL Q A

The 'replace' option is almost the same as the 'Find' option. There is an entry box to be filled in with the replacement text for the 'Find' text.

Find again

CTRL L

Repeat the last Find or Replace option unless the Global option was set. The replace option has an extra option in the "Options" box, "Prompt on replace" which will ask before each replacement.

Go to line number...

CTRL J L

Enter a line number in the text and then jump to this position.

Compile

Compile

F9

Compile the program ready to run in *Wildcat!* Just which file is compiled will depend on whether you have set a name in the following, "Primary file" option. With no primary file, the current window will be compiled into a *wcCODE* executable program with an extension of .WCX. If an error is encountered by the compiler, the cursor will be positioned under the offending item and a brief explanation displayed on the top or bottom row of the screen.

Primary file...

When you are using $INCLUDE files in your text you can nominate a primary file that is the main one you are working from. This means you can have an $INCLUDE file window open for editing and press the F9, compile key. The compile will begin with the primary file instead of trying to compile the current one which may not have sufficient information without the primary file.

Clear primary file

This option removes the primary file name from the compile chain.

Options

Set options...

This option allows you to change the maximum amount of memory to be used for editing each file in a window. When you run out of memory for a window the IDE will beep and allow no further editing. To increase the size, save and close all of the open windows, adjust the size and then re-open the files.

Window

Tile

Display all windows on the screen at the same time. The IDE will reduce the size of each window to try and fit them all on the screen.

Cascade

This will present the windows one on top of the other with each successive window being reduced in size by one line and column.

Close all

Pretty much that, close all of the open windows, prompt to save if the file has been modified.

Size

CTRL F5

Change the size and optionally move the active window. The SHIFT ↑ ← → ↓ keys will re-size the window and the ↑ ← → ↓ keys will move it. The process is terminated with the ENTER key, or aborted with the ESC key.

Zoom

F5

This will zoom the window to its maximum if it has been reduced, or to the reduced size if it is already at the full size. This can also be achieved with the mouse by clicking on the window-sizing arrow in the top right corner of the window.

Next

F6

When multiple windows are in use, this will step sequentially forward through each window and make it the active one.

Previous

SHIFT F6

When multiple windows are in use, this will step sequentially backward through each window and make it the active one.

Close

ALT F3

This will close the current active window.

List

ALT 0 (zero)

Display a list of all windows that are currently in use. You may then select a window to swap to by using ↑ ↓ followed by the ENTER key, or ESC to abort the process. Mouse users could use the scroll-bars, then select by double clicking on the name.

Help

The Help menu presents additional Help options. Certain keywords in many of the Help screens are highlighted in a contrasting color or screen attribute. You may click on any of these highlighted words in a help screen for a detailed explanation of the keyword.

Contents

This is a list of Help topics organized by menu function. You may select or click on any of these topics to view a help screen about that topic or keyword. Keyboard users can use their ⇥ (Tab) key to select and ENTER to view a Help screen.

Index

This is an alphabetic list of all keywords used by the *wcCODE* language and other help topics. You may select or click on any of these keywords to view a help screen about that keyword. Keyboard users can use their ⇥ (Tab) key to select, and ENTER to view a help screen.

About

This selection brings up a dialog box identifying the IDE version number and copyright information.

Let's move along now.

Well that about sums up the IDE. If you are a programmer with a good grasp of BASIC then you could leap ahead to Chapter three. In Chapter three we describe the keywords and their meanings for the *wcCODE* language.

If you are an enthusiastic, but budding programmer then the next section is for you. This will give you a basic understanding of what *wcCODE* can do for you and your BBS. If you have ever thought that programming was, or must be difficult, then scrap that idea and head for the next section where we will show you just how easy it really can be.

2 - Programming

What one knows is, in youth, of little moment; they know enough who know how to learn.

Henry Brooks Adams

In this chapter

The basic structure of *wcCODE* 43
General Program Information 44
When the over-view is over 50
Preamble to programming 51
- Line Numbers & Subroutines 52
- Pigeon Holes and Variables 54
- Keyboard Input 56
- Gentlemen, a little order please 60
- A Substantial Sub 66
- A Functional Function 70
- A Better File Method 72

We bid a fond farewell... 77

The basic structure of *wcCODE*

Now that we have explored the functions of the IDE, let's move on to more mundane things. If you are not all that familiar with the BASIC language, then just bear with us through this first section. Following this we have a quick-guide on programming in *wcCODE*, but first we need to mention things that will be necessary for experienced programmers to know about *wcCODE*.

If you are new to programming, you should read this next section anyway. You will want to refer back to it once you have mastered the basic skills of programming in *wcCODE*.

The limit to the length of lines in the IDE is 255 characters. If you prefer to scroll the screen left and right to edit your text then the IDE will happily accommodate you up to this limit.

The alternative to scrolling left and right is to use the backslash as a line-break character. The backslash must be the last character on a line for it to be considered as a line-break. As you are probably aware, the backslash is also used for Integer divide and the directory designator in Path statements. Because of this, you will need to ensure you do not use it in a command as the last character on the line, as it will be interpreted as the line continuation command.

Most times a program will fail to compile if a case is encountered where a backslash is mistakenly considered a line continuation. Therefore the possibility of such a problem occurring is extremely unlikely.

General Program Information

Every variation of BASIC is slightly different, and *wcCODE* is no exception. This section provides information about the various items that make up this language. It only deals with the underlying operational characteristics of the language, not the command set. The entire set of *wcCODE* commands are discussed in the following chapter.

BASIC internal data types

The following data types are those that *wcCODE* has in common with the BASIC language. There are other, *Wildcat!* -specific data types, that are fully outlined in the appendix in the back of this manual. For this section of the manual, we will confine ourselves to the structure of *wcCODE* that closely parallels the BASIC language. Once familiar with the constraints of *wcCODE*, then the use of the pre-defined data types in the appendix will be more obvious to you.

The data types

Type	Range	Format
Byte	0 to 255	8 bit unsigned
Integer	-32768 to 32767	16 bit signed
Long	-2147483648 to 2147483647	32 bit signed
Real	1.5e-45 to 3.4e38	48 bit internal Real
String	0 to 32767 characters long	variable length

Special Symbols

Besides letters, numbers, and spaces, *wcCODE* accepts the following special characters and pairs of characters and applies a specific meaning for the use of each. The math and conditional operators can be bracketed in parentheses to force the order of operation. If brackets are not used, then the outcome is calculated in the following order of precedence.

Operator	Notes
^	Exponentiation
*	Multiplication
/	Floating-point divide
\	Integer divide
Mod	Modulus
+	Addition
-	Subtraction
=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
And	Bit-wise logical And
Or	Bit-wise logical Or
XOr	Bit-wise logical exclusive Or
Eqv	Bit-wise logical Equivalence
Imp	Bit-wise logical Implication

Declarations

The keywords confined to variables, or identifiers declarations are **Dim**, **Const** and **Static**. These may be used one-per-line or variables may be assigned on the same line separated by commas. The **Static** identifier is generally only ever used inside a subroutine or function, but more on those two items later in the Chapter 3, *wcCODE* Command Reference. For now though, simply regard **Static** and **Dim** as working exactly the same.

```
Dim TestVar As Byte, BigByte As Long, TestStr As String
Const CV = "My life history", FuzzyBlock = Chr(176)
Static KeepCount As Byte
```

Numbers

The *wcCODE* language can accept numbers in

Decimal or Hexadecimal based formats.

Base	Identifier	Example
Decimal		41
Hexadecimal	&H (or 0x)	&H4F (or 0x4F)

Strings

A string is a sequence of zero or more characters from the extended ASCII character set. String literals must always be defined between sets of double quotes and can only be broken over line if the following conditions are met. If a string has to be broken over a screen line, then terminate it first with the quote character, followed by the concatenation character and then the backslash as the line-continuation character.

```
Dim ALongString As String
ALongString = "This is some program we are building, but it " + \
              "should be noted that this string will wrap at the" + \
              " end of the screen-width set in Wildcat!"
```

Strings can be declared with a limited length by using the asterisk and a numeric value to indicate the maximum length. Omitting the asterisk and the numeric value, allows the string to expand to the full maximum allowed by *wcCODE*, 32,767 characters.

```
Dim ANewString As String*50
TestString = "This is the string we know and love!"
ANewString = TestString
```

Keywords

The complete list of *wcCODE* keywords can be found in the appendix to this manual. These keywords cannot be used as identifiers. Since *wcCODE* is not case sensitive, these keywords may appear in any combination of upper or lower case in your programs.

Identifiers

Identifiers are the names that you assign to various structures, variables, data types, functions and procedures. They can be of virtually any length but a practical length of around 25 or 30 characters as a maximum should ease the debugging process. The choice however, is yours when it comes to the length of identifier names.

The identifier must begin with a letter of the alphabet and can only include letters, numbers and the underline character. As with keywords the case of the identifier is not important and a mixture of upper and lower case is quite acceptable.

```
Sub AnyThingGoesIn_Here(Prompt As String, Row As Byte, Col As Byte)
Dim A_Big_Array(120) = "Hello Dolly?"
```

Comments

Comments are notes to the programmer that are ignored by the compiler. They do not make any difference to the finished size of the compiled program nor do they affect its operating speed.

Comments are a very useful item when debugging code, so it's a good idea to include them as often as possible. You will be grateful you did during a later debugging session.

Comments are embedded in the source code by one of two processes. The apostrophe (') can be used to force the compiler to ignore everything following it. The alternative is the REM statement, which, when used as the first item on a line works exactly the same as the apostrophe. However, unlike the apostrophe, when REM is used at the end of a line of code, it must be preceded by a colon as a statement-end-marker.

```
Rem This will never be compiled.
' Nor will this
Print "Testing" : Rem Testing works, but this comment will escape
```

Constants

Constants can be declared to make the code easier to read. A constant cannot change its value once it has been declared in a **Const** statement.

```
Const LifeAndAllThat = 41
Const MoreYN = "Press [Enter] for more "
```

Fixed variables can be evaluated in the **Const** statement giving *wcCODE* another friendlier edge over BASIC. For example, in BASIC, to declare a constant value variable as a character, you need to be able to enter the character from the keyboard, but not so in *wcCODE*.

```
Const AFuzzyBlock = Chr(176)
```

Now let's see you do that in BASIC!

Subs and Functions

The use of subroutines and functions within *wcCODE* follows very closely to the way BASIC employs them, therefore their explanation is limited to the reference section in Chapter 3, *wcCODE* commands. Having said that, there is one very useful deviation that *wcCODE* Subs and Functions have from the BASIC language.

On occasions it can be useful to have a subroutine or function that will take a varying number of parameters in the argument list. The *wcCODE* language can handle this situation very elegantly, as long as two conditions are met when defining the Sub or Function.

The first condition is that the routine must be pre-declared using the "Declare" statement. The second condition applies to the parameters that are to be optional. These optional parameters must have their default value assigned in the declaration portion, but not in the definition of the actual routine. An example will serve far better here, than many lines of explanation.

```
REM The declaration of the routine to increment a count by one
Declare Function Inc(L As Long, I As Long = 1) As Long

REM The definition of the function
Function Inc(L As Long, I As Long) As Long
  Inc = L + I
End Function

Dim AVar As Long, BVar As Long
AVar = 12 : BVar = 56
Print  AVar; " + 1 = "; Inc(AVar); " and " \
       BVar; " + 16 = "; Inc(BVar, 16)
```

The default value of one, was supplied in this declaration for the increment to be used if none is provided when the function is called. If, instead a value is supplied, then that is what is used. This feature is not restricted to just one additional parameter, but any number may be added to the header information.

Types

The *wcCODE* language is a very strongly typed language. As well as the standard types outlined above, user-types can be built and assigned just as if they were any internal data type. The declaration of a user-type structure always begins with the keyword "Type" and ends with "End Type"

```
Type MyVeryOwnRecord
   MyName As String*30
   MyAge As Byte
   MySex As String*1
End Type
Dim MyRecord = MyVeryOwnRecord
MyRecord.MyName = "The fastest of the bunch!"
```

When the over-view is over

So much for a brief over-view of the *wcCODE* system. We have included some sample programs at the end of Chapter 3, *wcCODE* commands. These example programs can demonstrate the power of *wcCODE* at work. For now though, let's get down to developing some basic programming skills. If you are comfortable with programming then you might want to skip ahead to the Chapter 3, *wcCODE* commands. Although, as one ancient Greek philosopher put it, who was often quoted by the late Julius Sumner Miller, "There is no crime in not knowing, the crime is in not wanting to know!"

Preamble to programming

In this section you will get to see just a small part of what *wcCODE* is really all about. In order to automate a lot of your repetitive tasks in maintaining your *Wildcat!* BBS system, your knowledge of *wcCODE* will expand with frequent use. The items discussed here represent only a fraction of what is possible with *wcCODE* and in most cases the examples and instructions given are the "ideal" ones, rather than what must be adhered to.

You will find *wcCODE* an extremely powerful and flexible programming tool, making your life as a Sysop a little less onerous and certainly more enjoyable, once you master a basic sub-set of programming commands and approaches to making them work.

This section is not a tutorial on the *wcCODE* or BASIC language but rather an introduction to the ease of programming in *wcCODE*. You would be well advised to obtain a book on programming Microsoft's QuickBasic, if you are not familiar with the BASIC language and if anything looks confusing then consult the book. You can rest assured though, with a little bit of effort in the early stages you will pick up the programming process very quickly and be well on the way to happier BBSing.

The *wcCODE* language as has already been stated in the opening introduction to this chapter is very similar to the BASIC programming language. The BASIC language was developed over 25 years ago as a simple beginner's language and an introduction to programming computers. In this language a relatively small sub-set of keywords were assigned to perform the various tasks most often required by computer users. These keywords or reserved words, are words that the computer will interpret directly as commands and consequently perform the actions the programmer intended.

Given this set of keywords then, it was considered a simple matter to issue strings of commands by using them to control the computer's output and actions. However as mere humans, when we program computers the outcome is sometimes quite different to what we expected. It's often a case of "well that's what I meant, but not exactly what I wrote." In an

ideal world this would not happen, however in the real world of programming you can expect to encounter the occasional "bug" to mar your day.

A "bug" is your computer's way of telling you that your program does not work exactly as hoped for or designed. To meet this end, we have included WCIDE.EXE as a fully integrated workplace for you to create your programs, compile them and hopefully, sort out all of the errors and ambiguity before you finally install them in *Wildcat!*

Line Numbers & Subroutines

The original BASIC used line numbers for each line of code and as each line was encountered the computer would convert the entire line into machine language instructions that it could understand. When a specific task or chore needed to be performed by the computer, the flow of the program was directed to an absolute line number. On completion of the chore the flow was usually redirected back to another line number, often this was the one immediately following the first redirection.

While this approach worked, it suffered many shortcomings arising from lines being inserted or moved in the text to improve the program's actions. Often lines were deleted which would only be discovered when the program was running and a call was made to the now non-existent line number. This would usually result in crash and another bug to fix. If this deleted line was in a seldom used subroutine then it may take days or weeks before it was called. Subsequently the crash to high-light its existence, would not bring the computer to a grinding and messy halt until weeks or months after you had moved to another state.

So much for bugs and crashes. If every task in the program was only performed just the once and the source code was successfully debugged, then this system would not appear to be so bad. With the BASIC language as it then was, each line of code was taken individually and then converted into machine-language instructions that the computer could understand. After processing that line of commands, it would then take the next line, convert and process it until the end of the program was reached.

This worked fine until repetitive tasks were required and the computer had to convert each line every time the task or chore was called. This led to a process called compiling which converted the entire program into machine language instructions that the computer could operate on directly at full speed. Compiling the source code required an extra step in producing the program, but the increase in operating speed offset this additional requirement many times over.

Along with compiling came the abandonment of line numbers, although *wcCODE* will happily accept source code line numbering if you prefer. The current trend though, is for almost all incarnations of BASIC, is to do away with the line numbers and opt for a more structured or modular approach to the flow of the program.

We do not include any instructions or information on line-numbered programs in this manual, even though *wcCODE* will work happily with them.

Before we go any further we need to state a convention to be used in this manual. We have printed all of the *wcCODE* keywords in **Bold** letters. This has been done so you will know which ones are keywords and which are general programming words that you make up.

Okay, now back to basics. Where BASIC formerly had to perform a repetitive chore, called a subroutine, it was addressed by line number. For example a numbered subroutine would have required **GoSub** 1200 to branch off to line 1200, and at the end of the subroutine a **Return** instruction would direct the program flow back to the next instruction after **GoSub** 1200.

The convention now is to label the routine with a specific title following the keyword **Sub**, instead of line numbers. So using this naming convention the routine could now be named "**Sub** DoSomething" and ended by the commands **End Sub**. To access this subroutine your program now only need say "DoSomething" and it would be done, no matter where the actual subroutine resided in the source code.

This approach thus got around the problem of conflicting or deleted line numbers, as it no longer mattered where the subroutine was located. At

compile time, the location of the subroutine is stored for the computer to use at run time. One point that must be observed is that the subroutine must be positioned, or at least declared, (see **Declare** in the reference section) in the source code text before it is called by the working part of the program.

Pigeon Holes and Variables

In order to usefully handle information, the computer program needs some form of storage area where it can place names, numbers or general information for its own use. The term "variable" was assigned to this storage area and as this name clearly indicates, the information to be stored can be very variable.

There are two basic types of data a variable can be called upon to store and they are strings and numbers. The **Strings** generally contain anything that can be typed at the keyboard, which admittedly, can also include numbers. The difference between a number variable and a string variable is that the number variety can have math functions performed directly on them. The **Strings** on the other hand, need to be worked on a little in order to first extract the numeric information before you can perform math on it.

While there is really only one type of string variable, there are four types of numeric variables. These are organized by the amount of memory that each one requires to store the information. One other thing worth noting with the numeric variables is that the processing speed can vary for some of the types. Typically, **Byte** and **Integer** are the quickest to calculate and as the number of bytes of memory required increases, so does the time the processor takes to perform calculations.

So once again, here are the variable types available in *wcCODE*.

Type	Range	Format
Byte	0 to 255	8 bit unsigned
Integer	-32768 to 32767	16 bit signed
Long	-2147483648 to 2147483647	32 bit signed
Real	1.5e-45 to 3.4e38	48 bit internal Real
String	0 to 32767 characters long	variable length

In order to keep track of the information being stored, the variables are assigned unique names such as "FirstName", "lastName" or "Age" according to what is to be stored in them. This is akin to a set of pigeon holes with labels stuck to the edge of the box. Each time we get a user's First Name we would slip it into the pigeon hole marked, not surprisingly "FirstName"

These names are made up by you and should ideally, describe what they are holding in terms of data. Although you could easily call a variable "BrokenCartWheel" that could contain "A Rose by any other name..." or 41 or whatever you like. As long as it was of the right data type *wcCODE* would not care what it contained, but you will find it easier to program using meaningful variable names..

You will notice the use of capital letters in the variable name, BrokenCartWheel. Variables must consist of one-word names and may contain only letters, numbers and the underline character. The first character of any variable, must be a letter of the alphabet and then any combination of letters or numbers, for as long as you need to make it. However, limiting the length of variables to around 25 or 30 characters will make your debugging easier, as reading long variable combinations can become confusing.

The use of the capital letters is to make the name appear more readable to you, the programmer and does not affect the way in which the program works. If for instance you had a variable called "userstatusinfo" it all appears to make more sense if you type "UserStatusInfo" instead. An

alternative is to use the underline as well, so that "User_Status_Info" becomes even more legible with this treatment.

Keyboard Input

Having already mentioned that we can only store data of the correct type in each variable, regardless of what it is named, now would be a good time to demonstrate how this is achieved. So that *wcCODE* can allocate some storage space for the new variable we need to tell the compiler what we want to call it and what type of data we will be storing in it. To achieve this we use the **Dim** statement.

```
Dim FirstName As String
Dim Age As Integer
Dim PayAmount As Real
```

The "As" keyword is mandatory for all data types. The **String** data type has one more wrinkle to it that the other numeric types do not and that is the length specifier. We can use the length specifier to tell the compiler to allow space for only "x" many characters. Mostly, the use of the length specifier is optional, except in user-defined data types where it is mandatory. Don't worry too much about the "user-defined" stuff at this stage as we will get to that later, just remember that sometimes a length specifier can be necessary.

```
Dim FirstName As String*30
```

How to get data in and out of these variables? We could use the **Let** statement, which has been pretty much abandoned for direct assignment, which simply means tell the variable what it contains.

With formal BASIC the **Let** keyword is used...

```
Let FirstName = "Rick Heming"
Let Age = 43
```

By using the current convention of direct assignment, we can omit the **Let** keyword and just directly assign the value to the variable...

```
FirstName = "Jim Harrer"
Age = 37
```

You will notice the use of the quotes characters at each end of the string data we are assigning, these are mandatory for strings to indicate that they are in fact, strings.

To display the contents of a variable, we use the **Print** statement followed by the variable name. With the **Print** statement, it is quite legal to include as many variables on the one line as you need or prefer.

```
Print "The name is "; FirstName; " at age "; Age
```

Would print for each of the above examples...

The name is Rick Heming at age 43

The name is Jim Harrer at age 37

Notice the inclusion of spaces at the ends of the "The name is " and " at age " to separate the contents of the variables from the literal characters used in the **Print** statement. The semicolons following each statement tell the computer to print each one hard up against the previous one. A comma could be used here for spacing the statements out, but you can read about that in the reference section under the **Print** Statement.

So far so good. The variables so far, are usually referred to as simple variables and as the name implies they are fairly simple in that they will only store one piece of information in each pigeon hole. To service this shortfall, a data structure called an array is used. The array can be viewed as a series of pigeon holes in two dimensions. The first dimension is, as outlined for the simple variable, with an identical set following behind the first. The way we keep track of the next layer pigeon hole, or variable, is to give it an index number or subscript as they are called.

Typically, variable subscripts always start at zero, so the first label in the front set of pigeon holes for "FirstName" would be FirstName(0), LastName(0), Age(0). The set of pigeon holes immediately behind the first set, would contain FirstName(1), LastName(1), Age(1) and continue for as many sets of names you think you will require.

So with this knowledge at hand, in order to set up sufficient space for 10 FirstName, LastName and Age variables the **Dim** statement would be used as follows.

```
Dim FirstName(10) As String*30
Dim LastName(10) As String*40
Dim  Age(10) As Integer
```

What we are asking the compiler to do, is to allocate space for eleven lots of strings for the first names, eleven for the last names and eleven for the ages. In case you are wondering about the eleven, when we have only dimensioned for ten, then don't forget that these array subscripts automatically start at zero, which makes 11 in total. You can also instruct the compiler to allow exactly what you want with the **Dim** statement, by using the optional **To** keyword in the subscript designation.

```
Dim FirstName(1 To 10) As String*30
```

Now this would allow exactly ten subscripts, and the choice is yours whether you bother to assign your variables this way, or the previous way. The memory wastage is minimal by not declaring the lower bound subscript. If you were using a small number of subscripts, with unusually large values then perhaps you would do the following.

Dim SpecialArray(1201 **To** 1250) **As Integer**

The compiler would then only allocate enough space for the 50 integers you had asked for but the likelihood of a structure such as this would be extremely rare.

So let's get back into the flow of things again. To store the data in an array variable is little different from the previous example, we just need to say which element of the array we need to use.

```
FirstName(1) = "Scott"
LastName(1) = "Hunter"
Age(1) = 28
```

To print the data.

```
Print "My name is "; FirstName(1); " "; LastName(1); \
      " at age "; Age(1)
```

Okay so far? It also is perfectly legal to use a variable for the subscript. By the way when we say "legal" we mean a legal syntax in *wcCODE* and

not "legal" as in Cops and Robbers! Now back to variables as array subscript holders.

```
Dim Ctr As Integer
Ctr = 1
Print "My name is "; FirstName(Ctr); " "; LastName(Ctr); \
      " at age "; Age(Ctr)
```

This makes it easier to handle another of the *wcCODE* constructions and that is the **For...Next** loop for repetitive tasks. But first let's look at another way, an even better way of getting data into those variables.

While it was easy enough to add the information to the variable by enclosing it in quotes, that only allows you the programmer to enter it. What we really need is to be able to let the user of your program enter the information, so we take another of the *wcCODE* keywords, **Input**. The **Input** statement, as the name implies, allows input and this goes straight into your variables like this.

```
Input FirstName
```

When your *wcCODE* program is running, on encountering this statement it will wait while "stuff" is typed in at the keyboard. The program knows when the end of the data has been entered, because the user will press the ENTER key to say so. The above **Input** statement is a little crude, as in actual fact the user has no way of knowing what "stuff" to enter, so a better approach would be to combine both the **Print** and **Input** statements.

```
Print "Enter your first name: ";
Input FirstName
```

Notice the semicolon in the **Print** statement, it will leave the screen cursor at the end of the space following "...name: ", just so the user knows what to enter. Wow, we are really moving right along here but let's get back to that **For...Next** loop thing mentioned a little way back.

The **For...Next** construct is a handy device for all sorts of repetitive tasks. To use it, you merely set a start point and an end point and the rest is history. So let's gather some data and then print it back to the screen.

```
Dim Ctr As Integer
For Ctr = 1 To 10
    Print "First name: ";
    Input FirstName(Ctr)
Next Ctr
For Ctr = 1 To 10
    Print FirstName(Ctr)
Next Ctr
```

Now that little puppy will get ten first names and store them in memory for later use. How easy was that? To print them out was just as easy. Notice the indenting of the source code, to visually indicate *to you,* just where the loop structure starts and finishes. You would be well advised to develop an indenting method that looks right to you, for easier understanding and later debugging. It makes no difference to the compiler what your code looks like, so the clearer you make it now, the easier it will be to read in six month's time when you might want to make some changes.

So let's get a little serious now. As with later versions of BASIC, *wcCODE* can also expand on these two primary variable types, by using a structure called a record. A record is simply a method of grouping several of these variable types into a set structure, that keeps everything neatly together. The record structure is defined by using the keyword **Type**, followed by a name of the specific structure. Once again, this name is one you make up to describe what your record does or is used for.

Gentlemen, a little order please

Let's say you wanted to write an order questionnaire, for one of your BBS users to place an order for some of the fine products advertised on your board. Let's also assume that there are 40 different products you could take orders for. You would not want to assign separate variables for each line, which may require a product code, description, price, quantity and total as individual arrays.

Here's how we can streamline this by using a record structure.

```
Type OrderLineRecord
    Code As String*20
    Desc As String*30
    Price As Real
    Qty As Integer
    Tot As Real
End Type
```

Now, having developed our own data type, we can refer to it as if it were just like any of the other five types (Byte, Integer, Long, Real and String) that we have already discussed. We can use this new data type of our own-making in an array, or even within another data type that we also describe. Let's do that, as it sounds fairly complicated but is actually pretty straight forward to do.

```
Type OrderRecord
    Number As String*10
    Customer As String*30
    Line(1 To 25) As OrderLineRecord
End Type
```

Notice the string length designators, they are *mandatory* when used within these user-defined record structures. The compiler needs to allocate just enough space for them and therefore needs to know their length when you declare them.

Okay, so how do we access these new data types for getting the data into and out of them? Well, as you would expect of *wcCODE*, that's pretty easy too, we use the period or dot to define them.

```
Dim Order As OrderRecord, Ctr As Integer
Print "Order number: "; : Input Order.Number
Print "Customer name: "; : Input Order.Customer
For Ctr = 1 To 25
    Print "Product code: "; : Input Order.Line(Ctr).Code
    Print "Description: "; : Input Order.Line(Ctr).Desc
```

```
    Order.Line(Ctr).Price = InputNumber("Price: ", 0, 5000)
    Order.Line(Ctr).Qty = InputNumber("Quantity: ", 0, 5000)
    Order.Line(Ctr).Tot = Order.Line(Ctr).Qty * Order.Line(Ctr).Price
    Print "Total: "; Order.Line(Ctr).Tot
Next Ctr
```

Okay, now if you thought you were going well until now, when suddenly there's a colon sitting between the **Print** and the **Input** statement which no one had mentioned yet, then don't worry. That's BASIC's way of allowing more than one programming statement on each line. When the compiler encounters the colon, it is the same as if you had begun typing it on a new line.

Wow, check out the lines in the **For**...**Next** loop though. If that sent you reeling backwards, then claw your way back onto the chair and settle down. It isn't anywhere near as bad as it looks, even though it does look a little daunting. If you take it in sections, rather than the whole, it wont look so bad. What we are saying here is, **Input** to the "Order" variable, in the "line" array, subscript "Ctr", which is made up of another user defined variable, one item of which is the "Code". Easy huh?

Whoops! Someone turned the lines for the Price and Quantity around the wrong way. Well it might look like that but **Input** is a fairly generic form of gathering data from the keyboard but hey, this is *wcCODE* and we like to make life easier for you. What **InputNumber** does, is present your program's user with the prompt "Price: ", and then it waits for the user to type in only numbers.

The zero and the 5000 you see following the prompt tell **InputNumber** to ensure the numbers entered by the caller are in the required range.

The calculation of the order-line total, multiplying the Qty by the Price is pretty straight forward too, if you look at it. We don't need the user to work this one out, as that's what the computer is good at right? Number crunching.

So far we have attempted to fill the order but what happens if there are less than 25 lines to fill? We don't want the user to have to bang the

[Enter] key 25 times and get more frustrated with every time, so once again *wcCODE* to the rescue with **Exit**.

What we need to do, is perform some decision making during the filling in of the order form, so an **If...Then** statement looks needed here. Couple that with an **Exit** statement and we have a slick looking program already. So adding some extra logic we would slip this line in between the first and the second positions of the **For...Next** loop.

```
For Ctr = 1 To 25
   Print "Product code: "; : Input Order.Line(Ctr).Code
   If Order.Line(Ctr).Code = "" Then Exit For
   Print "Description: "; : Input Order.Line(Ctr).Desc
```

What this will do, is exit out of the **For...Next** loop as soon as the user presses the [ENTER] key on a blank line. The "**For**" is a mandatory form of the **Exit** statement, as you must tell the compiler just where you would like to exit from. There are several types of loops where the **Exit** command can be used and each one requires a specific instruction. It is quite acceptable to have loops within loops but more of that in the reference section under the heading of **Exit**.

Well so far, we have the customer's order but unless you are there to read it as it is typed in, then it needs to be stored somewhere. The most obvious place is on the disk as a file, so let's open a file and store this stuff in it.

```
Open "Orders.TXT" For Output As #1
```

That's all pretty self-explanatory really, we are asking to **Open** a file by the name of "ORDERS.TXT" that is created whether one exists by same name or not. We need to **Output** data to it, and for the time being we would like to refer to it as file **#1**. So how hard can it be to write the order out to the file? It isn't now that it is open, we use the **Print** statement. This time we tell **Print** exactly where we want it to go by referring to the file number **#1**. If you omit the **#1** the **Print** statement will think you mean the screen and place it there, not in the waiting file.

```
Print #1, Order.Number
Print #1, Order.Customer
For Ctr = 1 To 25
    Print #1, Order.Line(Ctr).Code
    Print #1, Order.Line(Ctr).Desc
    Print #1, Order.Line(Ctr).Price
    Print #1, Order.Line(Ctr).Qty
    Print #1, Order.Line(Ctr).Tot
Next Ctr
Close #1
```

So, we have written the data to the disk and issued a **Close** statement to tell the system we have finished writing to the file. Now, when you come along after a hard days work and need to brighten your day with a few cash orders, there is a bulging "ORDERS.TXT" file waiting to be processed. You will recall that when we wanted to get some data into our variables we used the **Input** statement. Well we used the **Print #1** statement to put the stuff into the file so logic being er, very logical, would suggest we use the **Input #1** statement to get it out of the file.

Correct again and here's how...

```
Open "Orders.TXT" For Input As #1
```

Not so hard to follow, we want to input the data so it makes sense that we open the file for **Input**. The next bit is identical to the statement that put the stuff in the file in the first place. The **Print** statement was replaced with the **Input** statement and again, after finishing, we issue a **Close** to say that's it, we're done!

```
Input #1, Order.Number
Input #1, Order.Customer
For Ctr = 1 To 25
    Input #1, Order.Line(Ctr).Code
    Input #1, Order.Line(Ctr).Desc
    Input #1, Order.Line(Ctr).Price
```

```
    Input #1, Order.Line(Ctr).Qty
    Input #1, Order.Line(Ctr).Tot
Next Ctr
Close #1
```

So that you can take heart and see the order, you just print it to the screen with a routine that should be getting very familiar by now. We have added a few extra prompts to remind you just what all this data means.

```
Print "Order number "; Order.Number
Print "Customer "; Order.Customer
Print "Code......  Description.......................  "+ \
      "Qty...  Total........."
Print String(40,"-")
For Ctr = 1 To 25
    Print Order.Line(Ctr).Code
    Print Order.Line(Ctr).Desc
    Print Order.Line(Ctr).Price
    Print Order.Line(Ctr).Qty
    Print Order.Line(Ctr).Tot
Next Ctr
```

Now that's looking pretty good, except for the **String**(40,"-"), what the hey does that mean. Didn't someone say that **String** was a data type, just a little while back? Well er, yes we did but then we have based *wcCODE* on BASIC remember. Now BASIC, God love it's little heart has this optional thing called **String** that is used to print a string of characters. If you're confused, then yell at someone up at Dartmouth where BASIC was first developed over 25 years back.

But what about that **String**(40,"-")? All that is saying is, print a string of the minus signs ("-") and make it 40 characters long. It sure beats holding down the minus key in a **Print** statement too.

Okay, so we now have an order but there has been a bag full of repetitive coding in some of the original input statements, so let's see if we can clean that up a bit. As you'll remember we said that *wcCODE* was powerful AND flexible, so lets get our hands dirty on some **Sub** and **Function** calls.

Now we get to the good bits, **Sub** and **Function**. This is where you can really produce some very usable code. What's more you can even make that re-usable code, to save you writing the same stuff over and over again. The **Sub** and **Function** functions are very similar in code construction but each has its own place in the scheme of things.

A Substantial Sub

We'll start with **Sub**, as that is pretty straight forward and is virtually identical to the coding we have demonstrated so far, with just a couple of minor wrinkles. The **Sub** statement is used to declare a subroutine or sub-program, so rather than bog down here with the description we'll demonstrate once again how easy this all is. So let's take the **Input** section of the orders program we began a little ways back. We will display it here again to save you thumbing back through the pages.

```
Dim Order As OrderRecord, Ctr As Integer
Print "Order number: "; : Input Order.Number
Print "Customer name: "; : Input Order.Customer
For Ctr = 1 To 25
   Print "Product code: "; : Input Order.Line(Ctr).Code
   If Order.Line(Ctr).Code = "" Then Exit For
   Print "Description: "; : Input Order.Line(Ctr).Desc
   Order.Line(Ctr).Price = InputNumber("Price: ", 0, 5000)
   Order.Line(Ctr).Qty = InputNumber("Quantity: ", 0, 5000)
   Order.Line(Ctr).Tot = Order.Line(Ctr).Qty * Order.Line(Ctr).Price
   Print "Total: "; Order.Line(Ctr).Tot
Next Ctr
```

Recognize it? Right, now looking at this a little, there are a few obviously repeated processes here. Namely the **Print** statement followed by the **Input** statement and this is a perfect example of where we can lean-down some of this code by using a **Sub**.

With a **Sub**, we can send it information and get back information and the **Sub** doesn't much care what we send, just as long as it conforms to what it is expecting in terms of data types. So we declare a **Sub** called, in this case "GetMyOrder" and this, like the variable names, is one you make up to describe what it is intended to do. You can call a **Sub** almost anything you like, with the exception of any of the *wcCODE* reserved words. However a descriptive name associated with the **Sub**'s actions will mean more to you when you are debugging your program. So here's the syntax.

```
Sub GetMyOrder(Prompt As String, InputStuff As String)
```

What we are saying here is: Let's create the **Sub** "GetMyOrder" with a string called Prompt, passed to it and send the input stuff back in another string called InputStuff, and life should be great. Now it might not be obvious which variable is for transferring data to the **Sub** and which one sends it back out again. The reason is that it doesn't matter with the heading or declaration of the **Sub**. What handles which is decided within the **Sub**'s code. So let's do that.

```
Sub GetMyOrder(Prompt As String, InputStuff As String)
   Print "Please enter the "; Trim(Prompt); ": ";
   Input InputStuff
End Sub
```

Now having written that we can then call the **Sub** like this...

```
GetMyOrder("Order number", Order.Number)
GetMyOrder("Customer name", Order.Customer)
For Ctr = 1 To 25
   GetMyOrder("Product code", Order.Line(Ctr).Code)
   If Order.Line(Ctr).Code = "" Then Exit For
   GetMyOrder("Description", Order.Line(Ctr).Desc)
   Order.Line(Ctr).Price = InputNumber("Price: ", 0, 5000)
```

```
    Order.Line(Ctr).Qty = InputNumber("Quantity: ", 0, 5000)
    Order.Line(Ctr).Tot = Order.Line(Ctr).Qty * Order.Line(Ctr).Price
    Print "Total: "; Order.Line(Ctr).Tot
Next Ctr
```

The only one that doesn't get included here is the **InputNumber** line as that's already been taken care of by *wcCODE* for you. Let's just go through what GetMyOrder is doing. Since all of the prompts need a colon and a space separating them from the data entered by the user, we can hang that on the end of the prompt, when we print it from within the **Sub**.

We've also smartened the prompt up by adding the "Please enter" part, and in case you were about to mention the **Trim** statement, that simply trims all of the blank spaces off the prompt before using it. Okay you say, there are no blanks spaces but remember, we are writing this **Sub** to be re-used many times over. At some time in the future, there just may be a few stray blank spaces which could make the prompt look a bit sloppy if they are not first trimmed out.

Our **Sub** could even get flashier than that and check to see if the colon already exists so we don't add another one on the end. Add a line in before the **Print** statement.

```
If Right(Trim(Prompt), 1) = ":" Then
    Prompt = Left(Trim(Prompt ), Len(Trim(Prompt)) -1)
```

What we are getting at here is: After **Trim**ming the blanks off both ends, check to see if the right-most character of the string called Prompt, is a colon. If it is a colon, then just copy everything up to, but not including the colon back into Prompt, so that when it is printed there is just the one colon at the end. Okay so let's talk about that Prompt = **Left(**.... section. The **Len(** section tells us the length of the Prompt after all of the blanks are trimmed off, so all we need are the left-most characters for the entire length of Prompt except the last one.

Unfortunately though, what will happen here is that the Prompt will be physically changed. When it is handed back after the user has pressed

the ENTER key and we may need that colon on the end later on in the program.

We could at this stage set a variable to say whether there was one there when the prompt arrived and add it back when the **Sub** is about to finish with it. However the better way, and you could almost bet we would have a better way, is to use a temporary variable that is defined within the **Sub**. We can discarded it when we have finished using it and that way we are not changing anything, so doing that, GetMyOrder now begins to look like this...

```
Sub GetMyOrder(Prompt As String, InputStuff As String)
  Dim TempStr As String : Rem A temporary string variable
  TempStr = Trim(Prompt) : Rem OK, so now we leave Prompt alone
  If Right(TempStr), 1) = ":" Then : Rem Check for a trailing colon
     TempStr = Left(TempStr), Len(TempStr) -1)
  End If
  Print "Please enter the "; TempStr; ": ";
  Input InputStuff
End Sub
```

Okay, we slipped in some **REM** statements in after the colon near the end of the line. The use of **REM** statements does not make any difference to your finished program, in terms of program size or operating speed, as they are simply ignored during the compiling process. The **REM** comments are notes to you, and it is a very good habit to get into using these liberally throughout your code. You may not think so at the moment, but six months later when you want to fine tune this **Sub**, you will be glad you took the time to add the comments.

Back to the strings now. Because the TempStr is dimensioned or defined within the **Sub** it is discarded as soon as the program flow leaves the **Sub**. The term "program flow" is regarded as the way each step of the program is accessed. After the user has pressed the [Enter] key, the program flow leaves the **Sub**. Any variables within the **Sub** are dispensed

with until the **Sub** is used again, when they are re-established and used with a completely new prompt.

The little order entry program is beginning to look a bit slick now, so let's add some more "bells and whistles" with a **Function** now that you are so obviously adept at handling the **Sub** statements. Right then, what is a **Function** and why is it needed if, as was mentioned some time back, it is similar to a **Sub**?

A Functional Function

A **Function** differs from a **Sub** in its ability to be assigned a value much in the same way a variable can be assigned a value, be it a numeric data type or a string. The **Function** can be a very powerful component of your program and it will be in your best interest to fully understand the use of these routines, especially when it comes to code re-usability.

So let's rip right into the **Function**. We will start off at the question we asked in the program to see whether the user had pressed the ENTER key on a blank product code to indicate that they were all finished with the order. We can streamline this a whole bunch by removing that line, adding a function and a new line to the original code. So let's do it.

```
Function YesAnswer(Prompt As String) As Integer
```

What is about to happen here is the **Function** is to be called "YesAnswer", we will get it to display the prompt whatever it may be and then return an integer value dependent on the user's response to the question.

```
Function YesAnswer(Prompt As String) As Integer
    Dim KeyIn As String*1
    YesAnswer = False : Rem Default to No just in case
    Do
        KeyIn = UCase(Inkey)
    Loop Until KeyIn > ""
    Print KeyIn : Rem Let the user see the key they pressed
    If KeyIn = "Y" Then
        YesAnswer = True
End Function
```

Okay so that doesn't look too serious, but let's go through it line by line. The first line we have declared a **String** variable called KeyIn, no surprises there. Next, and this is where the **Function** differs from a **Sub**, we can assign a value of **False** to this function called, "YesAnswer". You will notice the data type of "YesAnswer" is an **Integer**, but we have said it should equal **False**. Well in *wcCODE*, indicators like **True** and **False**, **On** and **Off** and the odd few others are integers with fixed values. The **True** value is equal to negative-one, and the **False** value is zero.

By setting the **Function** result to **False** straight off, it means that the only way the **Function** result can be **True** is if the user hits the letter "Y". Now this may not be that mission-critical in our order entry program, but it could wreak havoc if the user was answering a question to delete a batch of files for instance and they clipped the wrong key.

While on the subject of the letter "Y" let's explain a couple of things you haven't seen before, the **Do**...**Loop** and **Inkey**. The **Do**...**Loop** looks pretty straight forward and what it does is **Do** what ever is contained between it and the word **Loop**.

However **Loop** is not just an ordinary loop instruction, it has some strings attached (Pun fully intended there). Before it can loop back to the code immediately following the **Do** word, it must first check to see that our variable KeyIn is greater than nothing, say what? Okay we have used the ">" greater-than sign and two sets of quotes together. Two sets of quotes have nothing between them and KeyIn is a string right? So if KeyIn equals two sets of quotes, then KeyIn equals nothing and we are asking the program flow to **Loop** until KeyIn is greater than nothing.

Well that was simple enough and now what about **Inkey**? The **Inkey** function within *wcCODE* will read the keyboard for one very quick shot and if a key has been pressed then **Inkey** will grab it and put it wherever we have told it, in this case into KeyIn. Because **Inkey** is a one-shot look at the keyboard we need to have a loop for it to stay there until a key is pressed, therefore the **Do Loop** is needed. But what about this **UCase** stuff? you holler.

The **UCase** function is another of those things in *wcCODE* that will make your life easier, it converts a string of any length into all upper case letters. This means that your user doesn't bang the keyboard with frustration because your routine is expecting an upper case letter "Y" and they are hitting a lower case "y". Your program code might be looking for an upper case letter and ignoring the existence of the lower case version of the same letter.

No one said the computer was smart, it just means your code has to be smarter if you want to cut it as a programmer, so we convert the key stroke into an upper case version straight away to avoid the confusion. The **UCase** function has its counterpart **LCase** which does the opposite and converts a string into all lower case characters, if that's the way you prefer to work.

A Better File Method

So far we have now developed a pretty slick piece of programming but back a ways, when saving the data to the disk file, there were a lot of **Print** and **Input** statements that *wcCODE* can streamline. As well as the two methods of opening a file, **Output** and **Input** there is a third in this group called **Append**, and what **Append** does is, you guessed it, **Append** the next lot of data onto the end of the last lot. This is not just handy, this is mandatory if you get more than one order for products, and it would be hoped you did.

Both **Output** and **Append** will create a new file if the name you have provided does not exist but if the file does exist, then **Append** does not kill the contents as **Output** would. So a basic rule-of-thumb here might be to use **Output** when you know you want to create a new file and **Append** when you don't want to stomp on data that may already be sitting there on the disk.

So what about the better file method? Now we get really serious and introduce **Random** as a file opening method, with a mandatory length-of-record designator. You will recall we used a record structure for both the line item and the total order, when defining the variables a little earlier.

Well now we really put these records to work with the **Random** method of file handling.

```
Open "Orders.DAT" For Random As #1 Len = Len(OrderRecord)
```

So once again, one step at a time, let's look at what we have here. We are opening a file and this time it's been called "Orders.DAT", also this time we have changed the extension part of the file name to indicate that this not going to be an ordinary text or sequential file. We want this opened as a **Random** accessed file, which means we can bob around all over the place to find the data if we so choose. With the previous types of files we were restricted to a linear or serial approach to writing and then reading the data.

The **Random** file type overcomes this limitation. While not too big a deal in an order entry program, it could be very handy on a user database. Rather than have to step through every user until you located the one you want, **Random** would allow your program to store a key to each user's name and then use that key to jump straight to the right spot in the data file. This is where the **Len** = **Len** statement comes in as it is telling the system the size of each record that is to go into the file.

The **Len** = **Len** may look a little confusing but unfortunately that's the way of BASIC and we follow BASIC fairly closely, so the confusion must remain. The second use of the term **Len**, in the open-the-file statement will return the length of the data type, in this case called OrderRecord, which we defined earlier. As it turns out, the **Len** of OrderRecord is 1640 bytes so each record within the **Random** file will be this length too. When the system accesses the records in this type of file it will be regarded as complete records of data and not individual lines, as in the previous types.

Okay, so the **Random** file has a different structure. In order to move data to and from it, so we need to use a different method. The **Put** and **Get** functions are reserved for this chore and as they imply with their names, they put and get data to and from the disk. There is just one other item that needs to be mentioned. Because they are **Random** files of a fixed record length, they need a specific designator to position each record in the correct location in the file.

We use the term **Seek**, or alternatively, we can indicate in the **Put** or **Get** statement exactly where the record is to go. Once the user has finished filling in the order, we then send the data to the file with the **Put** statement in one single easy move.

```
Put #1, LOF(1) + 1 ,Order
```

Right, now take it easy here as there is nothing that really difficult. Okay, so no one warned you about this **LOF** thing but don't worry, as with the rest of *wcCODE*, **LOF** is easy too. The function that **LOF** performs, is to return a value for the Length Of File (LOF) and this value will be the current size of the data file in number of records. So if a new file has no records then **LOF** would return zero, for a file with ten records then **LOF** would return ten. We want to add our new record to the end of the file so we need the **LOF** of the file plus one to position the file pointer at the end.

We mentioned **Seek** a little ways back and we could have just as easily used it, as **Seek** does just as it sounds, it seeks out the position we are after.

```
Seek #1, LOF(1) + 1
Put #1,, Order
```

This would have done exactly the same thing as the previous command but notice that, if the file pointer position is left out of the **Put** statement, then there must still be a comma used as a place-holder for the missing instruction. When it comes time to read the data back, then we simply use the **Get** statement with a sequential **Seek** operation. Although, as a **Random** file, it need not be sequential but for the sake of the order entry program we will run though the data sequentially.

```
Sub PrintOrder
   Dim Ctr As Integer
   Print "Order number "; Order.Number
   Print "Customer "; Order.Customer
   Print "Code......  Description.......................  "+ \
         "Qty...  Total........."
   Print String(40,"-")
```

```
    For Ctr = 1 To 25
        Print Order.Line(Ctr).Code
        Print Order.Line(Ctr).Desc
        Print Order.Line(Ctr).Price
        Print Order.Line(Ctr).Qty
        Print Order.Line(Ctr).Tot
    Next Ctr
End Sub

Open "Orders.dat" For Random As #1 Len = Len(OrderRecord)
While Not(EOF(1))
    Get #1,,Order
    PrintOrder
Wend
Close #1
```

Uh-oh, just when you thought you had a handle on all of this, someone slips in a whole pile of stuff to confuse the issue again. But, as we said before, "don't panic, this is easy." Let's look at this one line at a time again.

The term "**While**" is fairly self explanatory here but perhaps, not so the gobble-de-gook following it. What we are getting at, is that **While** we are **Not** at the End Of the File (EOF) we called number one, then **Get** the record, pass it to the PrintOrder **Sub** and **Wend**. Wend, what is **Wend**? Beats me too, well not exactly, it stands for While **End** and that's the way BASIC demands it, so **Wend** it is. All this means is that, while the condition following the **While** statement is being met, then do whatever processes are contained between the **While** and the **Wend** statements.

Another point that may look a little confusing is the brackets used with the **Not** statement. In all logical statements, where conditions are evaluated back to a true or false outcome, the use of brackets is optional. It can be

a good programming practice to use brackets to indicate exactly the conditions you are trying to evaluate.

Including brackets around the EOF(1) statement in this case is superfluous, but some programmers always use brackets and others do not and it remains with you which you prefer. It makes virtually no difference to the resulting compiled program, but it can certainly make reading and debugging easier with their use.

We bid a fond farewell...

Well, programmer, this is where we leave you. By now the order entry program is really cookin' and we have shown you a whole bag full of functions and statements. So, armed with this small grab-bag of knowledge, you are ready to hit the keyboard and begin writing some applications of your own.

You can check out the example programs at the end of the reference section for more ideas on just how else you can use the *wcCODE* language.

Using the *wcCODE* Integrated Development Environment (WCIDE.EXE) you can create your code and compile all in the one area. If you have made a syntax error then *wcIDE* will notify you and place the cursor pretty close, if not exactly under the offending item. From there, you just make the necessary corrections and recompile until the compiler is happy with your work.

Having successfully compiled the program you will then need to test it thoroughly in *Wildcat!* Only there will you see whether the true fruits of your labors be lemons or cherries. Once it is fully tested, you can put your masterpiece on line, or better yet, even offer it for sale as a compiled program for other *Wildcat!* Sysops to use.

3 - Commands

People ask you for criticism, but they only want praise.

W. Somerset Maugham

In this chapter

wcCODE Command Reference 81

wcCODE Command Reference

The following pages contain a complete description of all of the available *Wildcat! 4*, *wcCODE* commands which are presented in the following format:

wcCODE Command

A brief description of the command.

Syntax

A description of the command's syntax. If the command is a Function then the data type it returns is shown in curly braces ({As DataType}) on the same line.

The use of parentheses around parameters are in most cases optional, with the following exception concerning **Function** calls. A **Function** with parameters, when used to return a value, always requires parentheses to bracket them. e.g. **If** Function(*String*) **Then**. While Statements with parameters do not show parentheses in the Syntax description, they can be used at your discretion. These rules will also apply to any **Function** or **Sub** that you define in your programs.

Often, the use of brackets when calling a **Function** or **Sub** can make the code more readable. It can visually group items as being attached to the **Function** or **Sub** name. The choice is yours as long as the rule for calling a **Function** as outlined above is observed.

Some **Function** results will be declared as "{As True/False}", but *wcCODE* does not have an internal Boolean data type. The Boolean values **True** and **False** in *wcCODE* are represented by negative-one and zero respectively.

A **Function** that returns true or false as a result, is actually of the integer type. To make the result of the function more obvious we have assigned "{As True/False}" to describe them, rather than their actual data type of "{As Integer}".

Remarks

This section contains a more detailed description of the command, including parameters and their meaning. Often this section will also describe how the command interacts with the rest of the *wcCODE* programming language and the system in general.

Note!

A special caution or comment that only appears in some Remarks sections. You really should read these notes even if you skip the Remarks section as the information contained here can often save a lot of frustration in your programming endeavors.

See also

Related *wcCODE* commands.

Example

Most examples will be a complete *wcCODE* program that you can type in, compile and run. All of the example programs are provided in a file called CODEFRAG.WCC. The code fragments can be cut and pasted from this file and compiled. However the entire file cannot be compiled due to conflicts in duplicate variable declarations.

Normally programmers develop a personal style or preference in the layout, appearance and construction of their source code. This manual has not followed this common approach. We have provided the example source code fragments with as many different constructs and styles as came to mind.

The intention was to indicate to new programmers, that a large proportion of the art is entirely personal. Providing the rules of syntax are observed, there are many ways of achieving the same end result with your programs.

Often the one line of code may contain two different ways of achieving the same outcome, neither may be *the correct* method. The *correct* method of writing any program is the way *you* prefer.

+ (Concatenation) Operator

Concatenate two or more strings.

Syntax

String1 + *String2* [+ *String3*...]

Remarks

This operator is used to combine several sub-strings into a single longer string. To concatenate several strings into one, they need not all be existing variables. During the concatenation process, other characters can be placed between the strings as literals enclosed in quotes.

In a string assignment, the Target string (the one to the left of the equals sign) must have sufficient room to accommodate the concatenation of all of the sub-strings including any literals. If the Target string has been declared with less length than that required for all of the strings to be concatenated, the result will be truncated to the maximum declared length of the Target string.

In some instances it can be better to use concatenation of strings rather than successive strings. Compile and run the second example to see the difference that concatenation can make.

Note!

If it is anticipated that a string is to be used as a Target string in a concatenation process then the length designator should *not* be used in the **Dim** statement for this string. This holds especially true if you are concatenating caller-entered data, as you often have little control over the string length they might enter.

See also

Left, Mid, Right

Example

```
Rem FIRST EXAMPLE
Rem Concatenate both literals and variables

Dim FName As String, LName As String
Print "Enter your first name ";
Input FName
Print " Enter your last name ";
Input LName
FName = "Hello " + Trim(FName) + " " + Trim(LName)
Print FName
Rem End of the main program body

Rem SECOND EXAMPLE
   Rem Successive statements using the semicolon
Print "The @Code color system in @"; "21"; "@ living color"
   Rem Concatenated statements using the plus sign
Print "The @Code color system in @"+ "21"+ "@ living color"
```

\ (Line-break) Instruction

Break a source code line.

Syntax

Remarks

This is not really an operator, statement or function but rather a feature of the compiler. The use of the backslash as the last character on the line, will instruct the compiler to assume the next line is still part of the current one. If that too, has a backslash the compiler will continue to append lines until a line without a trailing backslash is found, thus indicating the end of the entire statement.

The backslash character is also used for Integer divide and the DOS Path statement. Care should be taken to ensure that neither of these end up as the last character on a line, as the compiler will assume it to be a line break. Normally this should cause a compile-time error to highlight the problem.

If a literal string is to be broken, then the line should be terminated with a double quote, a plus sign then the backslash. See example.

Example

```
Rem Breaking long source code lines with the backslash

Dim YourLongVariableName As String
Const ShorterVariable = "Phew!! "
YourLongVariableName = " the use of the backslash"
Print ShorterVariable; "Now this is really getting silly"+ \
   " just to show"; YourLongVariablename;
```

ABS Function

Return the absolute value of a numeric value or expression.

Syntax

Abs(Number) {As Integer/Real}

Remarks

The result returned by the **Abs** function is the absolute value of the *Number* with any sign (plus or minus) removed. The value is not rounded either up or down for any fractional values, it is merely stripped of its sign bit.

The *Number* can be any valid numeric expression of type **Byte**, **Integer**, **Long** or **Real**. It will return the appropriate data type that was used when making the call.

See also

Fix, Int, Sgn

Example

```
Rem Two signed numbers with the same absolute value.

Dim IntA As Integer, IntB As Integer
IntA = +5 : IntB = -5
If Abs(IntA) = Abs(IntB) Then
   Print "IntA and IntB have the same absolute value"
   Print "And -5 multiplied by +5 appears positive! "; Abs(+5 * -5)
End If
```

ACTIVITYLOG Statement

Add a string to the activity log.

Syntax

ActivityLog *String*

Remarks

This command will add a string to the activity log for tracking a user's activity. The following example code could be included as an '$INCLUDE file with a call made to it for each *wcCODE* program a caller activates to track the frequency of usage of each one.

The activity log is a plain ASCII text file named ACTIVITY.###, where ### is the node number being logged. i.e. node 4's log would be ACTIVITY.4

Example

```
Rem Adding your own data to the activity log

Sub TrackMyProgs
    Dim DT As DateTime, tStr As String
    CurrentDateTime DT
    tStr = "Program: " + ProgName + " "
    tStr = tStr + FormatDate(DT.D,"mm/dd/yy") + \
                  FormatTime(DT.T," hh:mm")
    ActivityLog(tStr)
End Sub
TrackMyProgs
```

ADDFILE Function

Add a file to the Files database.

Syntax

AddFile(*FileRec*) {As True/False}

Remarks

The **AddFile** statement will add, as the name suggests, a file to the database. The information in *FileRec* must be at least, partially completed before **AddFile** will do its job. The minimum requirement for **AddFile** is a file's **Name** and the file area it is to be assigned to.

It is up to your program to ensure that sufficient information is added to the **FileRec** before the file can be added to the Files database.

Being a **Function**, **AddFile** will return a **True** or **False** result for your program to check for the success or failure of the operation.

See also

GetFileInfo, UpdateFile

Example

```
Rem Attempt to add a file to the database

Dim FRec As FileRecord, Ctr As Integer
Print "File name: "; : Input FRec.Name
Print "      Path: "; : Input FRec.StoredPath
For Ctr = 1 To 6
   Print "Keyword "; Ctr; " of 6 = ";
   Input FRec.KeyWords(Ctr)
   If FRec.KeyWords(Ctr) = "" Then Exit For
Next Ctr
If FRec.Name + FRec.StoredPath + FRec.KeyWords(1) > "" Then
```

```
    If AddFile(FRec) Then
        Print "File added OK"
    Else
        Print "AddFile error. File was not added!"
        Beep
    End If
End If
```

ADDMESSAGE Function

Add a message to the Message database.

Syntax

AddMessage(*MsgHeader*, *Message***)** {As True/False}

Remarks

The **AddMessage** function will add a message to the database into the currently selected conference. The *MsgHeader* is a variable of the internal data type **MessageHeader** and should be filled in with the appropriate message header details.

Your program should check that sufficient information is added to the header for it to be a relevant message in the database, before allowing it to be posted to the system. The message number is all that *wcCODE* will take care of automatically when adding a message. The **DestUserId** field only needs filling in if there is more than one user with the same name, then the unique User-Id must be used.

The *Message* part of this command will work in either of two ways. The first, it checks to see whether the contents of *Message* is a valid file name. If the file exists then that is regarded as containing the message text and its contents are added to the database.

If no file name is found matching the contents of *Message*, then the text *Message* contains is regarded as the message information to be added to the database.

If the message is directed to a specific user by name, and the **ToID** field contains a zero, *Wildcat!* will attempt to insert their Id from the database into the record. This will depend on two things, the user name must exist and there must be no duplicate user name in the database. If either of these conditions fail then a zero is placed in the **ToID** field of the record.

See also

GetMessage, UpdateMessage

Example

```
Rem Attempt to add a message to the database

Dim Msg As MessageHeader
Function FillMsg(MsgHd As MessageHeader) As Integer
   Dim Ok As Integer
   FillMsg = False : Rem Set a default value for the function
   Do
      DisplayPrompt(997) : Input MsgHd.From
      DisplayPrompt(990) : Input MsgHd.To
      DisplayPrompt(837) : Input MsgHd.Subject
      Ok = Trim(MsgHd.From + MsgHd.To + MsgHd.Subject) > ""
      If Not(Ok) Then
         If InputYesNo("Abort message ?") Then Exit Function
      End If
   Loop Until Ok
   FillMsg = True
End Function
Dim tStr As String
If FillMsg(Msg) Then
   Print "Message: ";
   Input tStr
   If tStr > "" Then AddMessage Msg, tStr
   Print "Message added"
Else
   Print "No message sent"
   WaitEnter
End If
```

ADDTIME Statement

Increase or decrease the caller's remaining time left on-line.

Syntax

AddTime [-]*Time*

Remarks

This statement will allow the addition or subtraction of the remaining on-line time for a user. This may be handy if a system event is pending then your program can reduce the displayed 'time remaining' prompt accordingly. Alternately the user could be winning or losing time in game.

Example

```
Rem Change a caller's time left on line with AddTime

Const Tword = "sillyanswer"
Dim AWord As String, Ctr As Integer : Ctr = 0
Do
    Ctr = Ctr + 1
    Print "Try ["; Ctr; "] Enter access code ";
    Input AWord
    AWord = LCase(AWord)
    If AWord <> Tword Then
        AddTime - 10
    Else
        AddTime + 10
    End If
Loop Until (Ctr = 3) Or (AWord = TWord)
```

ADDUSER Function

Add a new caller's details to the User database.

Syntax

AddUser(*UserRecord*) {As True/False}

Remarks

This statement will add a user's-entered-data-record to the database. The data must first be collected by your program using the pre-defined **UserRecord** data type. If the user was successfully added to the data base then this statement will return **True** otherwise it will return **False**.

For acceptance into the database, the new user information must at least contain a name in the **Name** field of **UserRecord**. If duplicate names are not allowed (set in *MAKEWILD*) then your program should ask the user to modify their name if the record is rejected.

See also

GetUser, UpdateUser

Example

```
Rem Add new data record to the user data base

Dim HotUser As UserRecord
Print "Enter the name of the new user ";
Input HotUser.Name
Print "Enter an alias if you use one ";
Input HotUser.Alias
If HotUser.Name > "" Then
   If AddUser(HotUser) Then
      Print "Name added OK"
      WaitEnter
   Else
```

```
        Print "Name was NOT added to the database!"
        WaitEnter
    End If
End If
```

AND Operator

Perform a logical And operation between operands.

Syntax

Op1 **And** *Op2*

Remarks

The *Op1* and *Op2* operands are logical expressions or numeric values of any valid data type of **Byte**, **Integer**, or **Long**.

The **And** operator may also be used for testing or comparing two or more conditions as a group. Parentheses may be used to group logical operations to make your code look more readable and to make the operators visually clear.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

Eqv, If, Imp, Not, Or, XOr

Example

```
Rem Perform a bit-wise logical manipulation on integers

Dim IntA As Integer, IntB As Integer : IntA = 129 : IntB = 4
If (IntA And 127) = 1 Then
   Print "IntA when ANDed with 127 does equal "; (IntA And 127)
End If
If (IntA > 30) And (12 < 100) Then
   Print "Both must be true!"
```

```
End If
If (IntA = 129) And (IntB = 4) And (IntA > IntB) Then
   Print "All three conditions have been met"
End if
```

ANSIDETECTED Function

Report whether the caller's terminal has ANSI capabilities.

Syntax

ANSIDetected {As True/False}

Remarks

The **ANSIDetected** function will return **True** if the caller's terminal has ANSI-terminal capabilities. This command may be used to determine whether ANSI cursor movement and color codes can be correctly handled at the caller's end.

This function reports what *Wildcat!* recorded on the state of the caller's ANSI capabilities during the logon process.

See Also

ANSIEnabled

Example

```
Rem Display color at the caller's end if it is possible

If ANSIDetected Then
   Print Chr(27); "[32m This should be colorful"
Else
   Print "No ANSI colors possible"
End If
```

ANSIENABLED Function

Report whether the caller has ANSI enabled.

Syntax

ANSIEnabled {As True/False}

Remarks

The **ANSIEnabled** function will return **True** if ANSI has been selected otherwise it returns **False**. This command determines whether the caller has elected to receive ANSI controls for color and cursor movement independent of any ANSIDetected result.

See Also

ANSIDetected

Example

```
Rem Set a display file defaults as appropriate

Dim FileExt As String*3
If ANSIEnabled Then
   FileExt = "SCR"
Else
   FileExt = "BBS"
End If
```

APPEND Declaration

Open a text or sequential file to append data.

Syntax

Open *FileName* For Append As #*FileNum*

Remarks

The **Append** declaration is an integral part of the **Open** statement for opening a file. Use **Append** where the file is to be created if it does not exist or extended if it does exist. The **Append** statement is only used for text or sequential files.

See also

Binary, Input, Open, Output, Random

Example

```
Rem Extend the data in your own tracking-file

Dim DT As DateTime
CurrentDateTime(DT)
Open "Tracking.LOG" For Append As #1
Print #1, "User: "; User.Name;
Print #1, "  When: "; FormatDate(DT.D,"mm/dd/yy  ");
Print #1, FormatTime(DT.T,"HH:MM")
Close #1
```

ASC Function

Return the ASCII value of a character.

Syntax

Asc(*String*) {As Byte}

Remarks

The *String* is any valid string expression which can also be a literal enclosed in quotes.

If the *String* length is greater than zero then the ASCII value of the first character will be returned within the range of 1 to 255. If the *String* length is zero then a value of 0 is returned.

This function is the counterpart of the **Chr** function.

See also

Chr, Val

Example

```
Rem Capitalize the first letter in a name if in lower case

Dim tStr As String*45
Input "Enter your first name "; tStr
Rem Check to see if it is already capped
If (Asc(tStr) >= 97) And (Asc(tStr) <= 123) Then
   tStr = Chr(Asc(Mid(tStr, 1, 1)) - 32) + \
            Mid(tStr, 2, Len(tStr) - 1)
End If
Print "Good afternoon "; tStr
```

ATLEN Function

Return the length of a string with the @Codes stripped out.

Syntax

AtLen(*String*) {As Integer}

Remarks

This function will return the length of the *String* with all of the *Wildcat!* @Codes stripped out. The string is not affected by this function call.

See also

Len

Example

```
Rem Check a file size, not including @Codes

Dim tStr As String, Ctr As Long
Open "HELP1.BBS" For Input As #1
While Not(EOF(1))
   Input #1, tStr
   Ctr = Ctr + AtLen(tStr)
Wend
Close #1
Print "The total character count for this file is "; Ctr; " bytes"
```

ATN Function

Return the arctangent of an angular numeric expression.

Syntax

Atn(*AngleVal*) {As Real}

Remarks

The argument *AngleVal* is an angle expressed in radians. Radians can be converted to degrees by multiplying by 180/pi where pi is approximately 3.14159.

See also

Cos, Sin, Tan

Example

```
Rem Compute and display the value of pi.

Print "The value of pi is roughly "; 4*Atn(1)
```

BAUDRATE function

Return the current connection baud rate.

Syntax

BaudRate {As Long}

Remarks

This function returns the current connect Baud rate as type **Long**. If the user is connected locally then zero is returned.

Example

```
Rem Keep track of high-speed modem users

Dim tStr As String
If BaudRate >= 9600 Then
   tStr = Str(BaudRate)
   Open "FastUser.LOG" For Append As #1
   Print #1, User.Name; " Logged at "; tStr
   Close #1
End If
```

BEEP Statement

Sound the beeper.

Syntax

Beep

Remarks

Sound the system bell or beeper. This will send the **Beep** command to the caller's system as well as the local console.

See also

Sound

Example

```
Rem Sound a warning on error.

Dim NameStr As String, TryCtr As Integer
TryCtr = 0
Do While (Len(NameStr) = 0) And (TryCtr < 3)
   Print "Enter your name "; : Input NameStr
   TryCtr = TryCtr + 1
   If TryCtr > 3 Then
      Print "Three strikes and you're out!"
      Beep : Chain "GetOff.WCX"
   End If
   If Len(NameStr) = 0 Then
      Print "You must enter your name here"
      Beep
   End If
Loop
```

BINARY Declaration

Open a file for binary access.

Syntax

Open *FileName* For Binary As #*FileNum*

Remarks

The **Binary** declaration is an integral part of the **Open** statement for opening a file as single byte record length file. The use of **Binary** will create a file if it does not exist. Use **Binary** to access a file randomly using **Seek**, **Get** and **Put** for the data transfer. The data contained in a **Binary** file can be of any byte value from 0 through 255 inclusive. The record length for any **Binary** file is one byte.

See also

Append, Input, Open, Output, Random

Example

```
Rem Display the contents of a binary file

Dim tByte As Byte, Ctr As Integer
Open "WILDCAT.EXE" For Binary As #1
For Ctr = 1 To 256
   Get #1, Ctr, tByte
   If (Ctr > 1) And ((Ctr Mod 16) = 0) Then
      Print Right(Hex(tByte), 2)
   Else
      Print Right(Hex(tByte), 2); " ";
   End If
Next Ctr
Close #1
```

BULLETIN Statement

Display a specific bulletin file.

Syntax

Bulletin *Num*

Remarks

This statement will display the contents of the *Num* bulletin if it exists. If the *Num* bulletin does not exist then a bulletin error message will be displayed.

Example

```
Rem Display the mandatory bulletin

Print "Welcome. As a new user you must at least read the "
Print "Rules and Manners bulletin."
GluttonForPunishment:
Bulletin 1
If InputYesNo("Read it again? ") Then
   GoTo GluttonForPunishment
End If
```

BULLETINMENU Statement

Show the bulletin menu.

Syntax

BulletinMenu

Remarks

This statement will display the Bulletin menu exactly as if the user had selected the [B]..Bulletin menu option.

See also

CheckBulletins, CheckMail, CheckMailAtLogon

Example

```
Rem Present the caller with the Bulletin menu

If InputYesNo("Jump straight to the bulletins? ") Then
   BulletinMenu
End If
```

BYTE Type

An internal data type for storing one byte values.

Syntax

Dim *Name* As Byte

Remarks

Variables of type **Byte** can hold values in the range of 0 to 255.

Byte variables are unsigned and therefore cannot accept a negative value. Assigning a value greater than 255 to a **Byte** variable will not declare an error but the resulting value will be equal to the right-most 8 bits of the value passed. For example a value of 257 passed to a **Byte** variable will return a value of 1.

See also

Dim, Integer, Long, Real, Type, Const

Example

```
Rem Assign a Byte-variable a value.

Dim BigDeal As Byte, BiggerDeal As Integer
BigDeal = 205
Print "This Byte value is "; BigDeal
BiggerDeal = -400
BigDeal = BiggerDeal
Print "Now look at this for a Byte value "; BigDeal
```

CALL Statement

Execute a subroutine or function.

Syntax

Call Name [*Arg* [,*Arg* ...]]

Name [*Arg*[, *Arg* ...]]

Remarks

The *Name* is the name of the subroutine or function to execute.

Before any subroutine or function can be called they must be fully completed or at least **Declared**.

The *Arg* is an argument or arguments that are passed to the subroutine or function with multiple arguments separated by commas. These arguments are also referred to as parameters.

The use of the **Call** statement is optional and it's inclusion is only to maintain compatibility with the BASIC language on which *wcCODE* is based.

The use of the parentheses is usually optional for bracketing the parameters, but be aware of the difference between reference and value when using parentheses. The only time when bracketing is not optional, is when the string being passed to the **Sub** or **Function** has been declared with a fixed length in the **Dim** statement.

By default, parameters are normally passed to the subroutine or function by reference. This means the subroutine can change the original value of each of the *Args*. This can cause debugging headaches when a subroutine or function uses a parameter and changes it before passing it back.

Passing by reference conserves memory as the actual variable passed as a parameter is directly referred to and manipulated by the subroutine. When passing by value, the subroutine creates a new variable to work with that leaves the passed parameter untouched.

Passing by value is achieved by either passing an expression (e.g. IntA+1) or placing parenthesis around the individual argument. This will then ensure the original is left untouched by the call to the subroutine or function.

See also

Declare, Function, GoSub, Sub

Example

```
Rem Set up a Sub and Call it without Call

Sub CapName(tStr As String)
   tStr = LCase(tStr)
   tStr = UCase(Left(tStr, 1)) + Mid(tStr, 2, Len(tStr) - 1)
End Sub

Dim TestName As String
Print "Please enter your first name "; : Input TestName
CapName((TestName)) : Rem The least ambiguous approach
Print "By value "; TestName
CapName TestName : Rem Valid, but prone to oversight and confusion
Print "By reference "; TestName
```

CALLERIDRESULT Function

Return the caller ID information.

Syntax

CallerIdResult {As String}

Remarks

The **CallerIdResult** as the name implies, will return the callerID as passed to *Wildcat!* by the modem. If caller-ID is available on your local phone network and your modem supports it, then *Wildcat!* will pass the information without modification to this function call.

Example

```
Rem Store the caller's phone number in a file

Dim DT As DateTime, tStr As String
Open "Caller.LOG" For Append As #1
CurrentDateTime(DT)
tStr = "Caller: "
tStr = tStr + FormatDate(DT.D,"mm/dd/yy  ") + \
              FormatTime(DT.T,"hh:mmTE  ") + \
              CallerIdResult
Print #1, tStr
Close #1
```

CAPTURE Statement

Open or close a capture file.

Syntax

Capture {On | Off}

Remarks

The **On** switch turns on the capture file named Capture.### where ### is the caller's node number. *Wildcat!* will append data to this file if it exists or create it if it does not and will continue to **Capture** until it is turned off by your program or the caller hangs up.

The **Off** switch closes the currently open capture file. If one is not currently open then no error will be declared.

Example

```
Rem Turn Capture On if required

Dim CapOn As String*1
Print "Capture all data for the Sysop? "; : Input CapOn
   Rem You can use either Asc or direct a character
   Rem as Asc(89) = the uppercase letter "Y"
If (Asc(CapOn) = 89) Or (CapOn = "y") Then
   Capture(On)       : Rem ASCII "Y" = 89
End If
```

CARRIER Function

Report the state of the line carrier.

Syntax

Carrier {As True/False}

Remarks

The Carrier function returns True if the modem reports that it is on-line and connected to another modem, if not then it returns False.

See Also

CarrierCheck, GoodBye, HangUp

Example

```
Rem A basic shell for almost all communication.

CarrierCheck(Off) : Rem Let's do this ourselves
While Carrier = On  : Rem The '= On' is optional as 'On' is implied
   Rem The bulk of your program in here in case the user
   Rem drops carrier at their end or the line drops out.
Wend
```

CARRIERCHECK Function

Turn automatic carrier detection on or off.

Syntax

CarrierCheck({On | Off}) {As True/False}

Remarks

Calling this function without parameters returns the current state of the CarrierCheck.

The CarrierCheck statement when turned On will monitor the line status and if it drops out or the user hangs up the program will terminate at that point. This would be the preferred option for most of your programs but there may be some occasions where you want the program to continue even after the caller has hung up.

By setting CarrierCheck to Off the program will continue until its normal termination point. This facility could be used as a call-back verifier or a logging device after the caller has tried an illegal access and been disconnected with the HangUp or GoodBye command.

See also

Carrier, GoodBye, HangUp

Example

```
Rem The basis of an after logon, call-back system

Dim PhoneNum As String
PhoneNum = User.DataNumber
Print "Stand by, we will now call you back"
CarrierCheck Off : HangUp
Delay 10 : Rem Wait for things to settle
Send "ATZ" + Chr(13)
Send "ATDT " + PhoneNum + Chr(13)
Delay(30) : Rem Hopefully long enough to establish carrier
CarrierCheck On
```

CASE Statement

Used in the Select Case statement.

Syntax

Case *CaseCondition*[, *CaseCondition* ...]

Statements...

Remarks

The **Case** keyword is only used within the **Select Case** statement.

The *CaseCondition* can be a numeric or string expression, a numeric range using the **To** keyword or a Boolean (True / False) condition.

The *Statements* are any valid *wcCODE* programming statements which can also be calls to subroutines or functions. The *Statements* are grouped as for normal *wcCODE* program statements and this block can be as many lines in length as required. The *Statements* portion is terminated when the compiler encounters the next **Case** *CaseCondition*, **Case Else** or the **End Select** keywords.

An additional case selector of **Case Else** can be used as a 'catch-all' for any conditions that are not met by any of the prescribed *CaseConditions*. The use of **Case Else** is optional within the **Select Case** statement.

See also

Else, End, If, Select Case

Example

```
Rem Get a transmission protocol from the caller.

Dim TransProt As String*1, Protocol As String*6
while Len(Protocol) = 0
   Print "Enter [Z]-ZModem  [K]-Kermit  [Q]-Quit (ZKQ) ";
   Input TransProt
```

```
    Select Case UCase(TransProt) : Rem Force upper case
        Case "Z"
            Protocol = "ZModem"
        Case "K"
            Protocol = "Kermit"
        Case "Q"
            Protocol = "None"
        Case Else
            Print "That was not in the selection."
            Beep
    End Select
Wend
```

CATCH Statement

Catch runtime errors.

Syntax

Catch *ErrValue* [, *ErrValue*...]

Catch All

Remarks

The **Catch** mechanism provides a convenient way of responding to errors which may occur while your compiled program is running.

Any **Catch** statement must be placed at the *end* of a user defined subroutine or function within the **Sub** *Name* ... **End Sub** framework, or at the end of the main program body. During normal operation when no runtime error occurs all statements after the **Catch** keyword are ignored.

If a runtime error occurs the **Catch** block for the currently executing subroutine or function will be searched for a handler for the specific error. If there is no **Catch** block or specific **Catch** handler for the error then control will return to the calling subroutine or function. The calling subroutine or function and its **Catch** block (if any exists) will then be searched for a suitable handler. This process will continue back along the entire **Call** chain.

If no appropriate **Catch** block or handlers are encountered along the entire **Call** chain the error passes all the way up to the main program body. Once there, if there is still no suitable **Catch** handler for the error then the program is terminated and the corresponding error message is displayed.

If **Catch All** is used then all non-defined errors will be handled but *Wildcat!* will terminate the program process as if it had ended normally. To overcome this situation after a **Catch All** statement, a recovery file could be invoked with the **Chain** command. As an alternative, a **GoTo** statement could redirect program flow to a re-start label within your code.

The *ErrValue* values must be one of the following:

ERR_ARRAYSUBSCRIPT

This error is caused by attempting to access an array using an invalid subscript index.

ERR_FILEALREADYOPEN

This error is caused by using the same file number to open another file. The file number or handle is already in use as a result of a prior call to the **Open** statement with the same file number.

ERR_FILEOPEN

This error is caused during a file opening operation.

ERR_FILERENAME

This error is caused by an error during a file re-name operation using the **Name** statement.

This error could be produced by...

- The original file not being found.
- The file is already open and in use by another application.
- The destination file name already exists.

ERR_INVALIDFILENUMBER

This error is caused by using an invalid file number in any file operation including **Close**, **Input**, **LOF**, **Open**, **Print**

ERR_MATH

This error is caused by trying to divide by zero, taking the logarithm of zero or a negative number, or by taking the square root of a negative number.

ERR_PATH

This error is caused by an invalid drive or path name in either **MkDir** or **RmDir**.

ERR_RUN

This error is caused when the file name passed to the **Run** command could not be found, or there was a problem encountered trying to run it.

ERR_TIMEOUT

Caused by a timeout during the **Input #***FileNum* command.

See also

Error

Example

```
Rem Use 'Catch' and 'Catch All' statements.
Rem Save this as 'ReadFile.WCC'. Compile and
Rem use it in 'Chain' example

Sub OpenForInput(FNameStr As String, AllOk As Byte)
   AllOk = True : Rem Set this to TRUE just to start with
   Open FNameStr For Input As #1
   Catch Err_FileOpen
      Print "Could not find the file ["; FNameStr; "] to open"
      AllOk = False
End Sub

Dim FNameStr As String, DataStr As String, FileOk As Byte
Print "Read which file name "; : Input FNameStr
If Len(FNameStr) > 0 Then
   OpenForInput(FNameStr, FileOk)
   If FileOk Then
      While Not(EOF(1))
         Input #1, DataStr
```

```
        Print DataStr
    Wend
    Close #1
  Else
    Print "Halting as file "; FNameStr; " could not be read"
    Beep : Delay 3
  End If : Rem The file was opened successfully
End If : Rem A suitable length file name was supplied
Catch All
  Print "An undefined error occurred. Process halting."
  Delay 3 : Rem Show this message for 3 seconds
```

CHAIN Statement

Transfer control to another compiled *wcCODE* program.

Syntax

Chain *FileName*

Remarks

This command terminates execution of the current program and starts execution of the compiled *FileName* program. All currently open files are closed and all variables are discarded. Passing variable values to the **Chain**ed program may be accomplished by using an intermediary data file created by the calling program.

The *FileName* file, must exist in the current directory or a full drive and path must be supplied. The *FileName* must be a compiled *wcCODE* program . The extension of .WCX is forced at run-time, so the program to **Chain**, must have a .WCX extension.

A compiled *wcCODE* program file called by the **Chain** command is no different from any other compiled *wcCODE* program that could be run successfully as a stand-alone program.

Note!

This a run-time only statement and *FileName* is *not* automatically included and compiled during the compilation process by this command. The first your program knows about the existence of *FileName* is when it is called upon to run. If this **Chain** command is in an infrequently used subroutine or function any errors in chaining the file may not be found for some time.

See also

End, $Include, Run, Stop

Example

```
Rem Use ReadFile.WCX program defined in 'Catch'
Dim DoWhatStr As String*1
Rem The following is a label for the GoTo statement
RetryEntry:
Print "Enter [R]-Read file  [Q]-Quit to menu (RQ) ";
Input DoWhatStr
Select Case DoWhatStr : Rem Upper/lower covered next line
   Case "R", "r"
      Chain "ReadFile.WCX"
   Case "Q", "q"
      Print "Returning to the menu "; : Rem Leave cursor here
      Delay 10
   Case Else
      GoTo RetryEntry
End Select
```

CHANGESETTINGS Statement

Change the caller's default settings.

Syntax

ChangeSettings

Remarks

This is command provides the user with exactly the same response as if they had selected the [Y]..Your settings menu option.

Example

```
Rem Offer the caller the option of changing their settings

If InputYesNo("Change your default settings? ") Then
   ChangeSettings
End If
```

CHECKBULLETINS Statement

Check for new bulletins.

Syntax

CheckBulletins

Remarks

This statement will check to see if there are any new bulletins for the user and if so it will present the normal bulletin prompt options. If there are no unread bulletins for this user then it will continue without presenting anything.

Note!

This command is intended to be used in a logon program and as already noted it will not return any information if there are no unread bulletins for the user. If you need your program to have the user check for bulletins by making a menu selection then use the BulletinMenu statement to present the entire bulletin options.

See also

CheckMail, CheckMailAtLogon, CheckNewsLetter

Example

```
Rem A very basic logon program

QuoteOfTheDay
CheckNewsLetter
CheckBulletins
CheckMailAtLogon
```

CHECKMAIL Statement

Check for a caller's mail.

Syntax

CheckMail

Remarks

This statement will check for any new mail for this user since they last called. If any is found then the messages will be listed, otherwise the user is notified there is no mail and an ENTER key response is required to continue the program.

While this statement appears identical in operation to CheckMailAtLogon the only difference is that CheckMailAtLogon will offer the choice of reading the mail, whereas CheckMail will just list the new mail.

See also

CheckBulletins, CheckMailAtLogon, CheckNewsLetter

Example

```
Rem A very basic logon program

QuoteOfTheDay
CheckNewsLetter
CheckBulletins
CheckMail
```

CHECKMAILATLOGON Statement

Check for caller's mail at logon and wait for a response.

Syntax

CheckMailAtLogon

Remarks

This statement will check for any new mail at the logon phase for this caller since the last call. If any is found then the messages will be listed and the caller given the opportunity to read the mail. If there is no new mail then the caller is notified there is none and an ENTER key response is required to continue the program.

While this statement appears identical in operation to **CheckMail** the only difference is that **CheckMailAtLogon** will offer the choice of reading the mail, whereas **CheckMail** will just list the new mail.

See also

CheckBulletins, CheckMail, CheckNewsLetter

Example

```
Rem A very basic logon program

QuoteOfTheDay
CheckNewsLetter
CheckBulletins
CheckMailAtLogon
```

CHECKNEWSLETTER Statement

Check for a new newsletter since the callers last logon.

Syntax

CheckNewsLetter

Remarks

This statement will check for a new newsletter. If no new news is good news then nothing is returned.

Note!

This command is intended to be used in a logon program and as already noted it will not return any information if no news letter exists.

See also

CheckBulletins, CheckMail, CheckMailAtLogon

Example

```
Rem A very basic logon program

QuoteOfTheDay
CheckNewsLetter
CheckBulletins
CheckMailAtLogon
```

CHR Function

Return the ASCII character for the supplied value.

Syntax

Chr(*Val*) {As String}

Remarks

This function returns the ASCII character corresponding to *Val* and is the opposite of the **Asc** function. The range of printable ASCII characters is between 0 and 255. For more information on the ASCII characters that can be generated using this function refer to the appendix in this manual.

See also

Asc, Space, String

Example

```
Rem Force a user entered character to upper case

Dim YesSir As String*1
Print "Ready to Rock and Roll: "; : Input YesSir
If (Asc(YesSir) >= 97) And (Asc(YesSir) <= 122) Then
   YesSir = Chr(Asc(YesSir)-32)
End If
Print YesSir
```

CLOSE Statement

Close a file or all files.

Syntax

Close [[#]*FileNum*[, [#]*FileNum*]...]

Remarks

The **Close** statement when called with a *FileNum* or sequence of *FileNum* separated by commas will close the file or files designated. When **Close** is called without any *FileNum* it will close all opened files.

The use of the hash or pound symbol '#' preceding the *FileNum* file number designator is optional with **Close** statement.

Although *Wildcat!* will automatically close files when the program terminates, you should always **Close** any files that you open as soon as you are finished using them for input or output. This is especially true for files opened for output as a **Close** command will force a write of any or all data currently held in internal buffers ensuring your data makes it to the disk in case of a hardware failure.

See also

Open, Reset, Input, Print

Example

```
Rem Open, Copy a text file and Close both

Dim tStr As String
Open "MyCopy.TXT" For Input As #1
Open "YourCopy.TXT" For Output As #2
While Not(EOF(1))
   Input #1, tStr : Print #2, tStr
Wend : Close #1 : Close #2
```

CLREOL Statement

Clear from the cursor to the end of the current screen-line.

Syntax

ClrEOL

Remarks

Clear the screen line from the current cursor position to the end of the line. This statement will clear the character under the cursor rather than starting at a position immediately to the right of the cursor.

The ClrEOL statement can be used to clear an entire line if the cursor is positioned at column 1, the left-most column of the screen.

See also

Cls

Example

```
Rem Clear a line to print a prompt

Sub ShowPrompt(Row As Byte, Col As Byte, Prompt As String)
   Locate Row, Col
   ClrEOL
   Print Prompt;: Rem Keep the cursor at the end
End Sub

Cls
Locate 23, 1
Print "This will not last long in this "+ \
      "position as it waits 2 seconds...";
Delay(2)
ShowPrompt(23,11,"NOW clear the line")
WaitEnter
```

CLS Statement

Clear the entire screen.

Syntax

Cls

Remarks

The Cls command will clear the screen, position the cursor in the top left corner and reset the current text color to the default.

See also

Color, ClrEOL

Example

```
Rem Clear the screen

Cls
Print "Starting on line one"
WaitEnter
```

COLOR Statement

Set the current terminal color.

Syntax

Color *Foreground* [,*Background*]

Remarks

Sets the current screen color to the specified *Foreground* value and optionally the *Background* value if it is supplied. If a background value is not specified then it remains unchanged.

For foreground colors the values 0 through 7 display a normal intensity with 8 through 15 displaying in high intensity. The background colors use 0 through 7 for still or steady colors and 8 through 15 for blinking characters but with the same color background as for 0 through 7.

Note!

For a table of all of the colors used by *Wildcat!* refer to the Appendix in the back of this manual.

See also

Cls, Screen

Example

```
Rem Display all the colors and their values

Dim FG As Integer, BG As Integer
Dim Row As Integer, Col As Integer
Col = 1
Cls
MorePrompt Off : Rem Stop Pause interfering
For FG = 0 To 15 : Rem Print a column of values
   Print FG
```

```
Next
For FG = 0 To 15
    Col = Col + 4 : Rem Position out to the next column
    Row = 0 : Rem Reset the Row position for each column
    For BG = 0 To 15
        Row = Row + 1
        Color FG, BG
        Locate Row, Col
        Print FG
    Next BG
Next FG
```

COMMENT Statement

Leave a comment to the Sysop.

Syntax

Comment

Remarks

This statement will prompt the user for leaving a message to the Sysop in exactly the same fashion as selecting the [C]..Comments to the Sysop menu option.

Example

```
Rem Allow the user to leave a message

If InputYesNo("All finished for today? ") Then
   Comment
   If Not(Local) Then HangUp
Else
   If Not(Local) Then GoodBye
End If
```

CONST Statement

Used to assign symbolic names to represent values.

Syntax

Const *Name* = *Val* [, Name = *Val*...]

Remarks

The *Name* is a meaningful label you make up to identify a specific value and *Val* is the value assigned to it. Constants can be declared on a single line separated by commas or they can be declared individually in which case the **Const** statement must be used at the beginning of each line.

The data type of *Name* is determined by the type of the *expression*.

Constants defined in subroutines or functions are automatically declared local and can only be used within that subroutine or function. These constants are virtually discarded when program flow leaves the subroutine or function.

Constants cannot be changed at any time or anywhere in the program.

Constants can be declared with direct values or assigned values as in the example below for the FuzzyBlock.

See also

Dim, Static

Example

```
Rem Typical Constant declarations

Const LocalBBSName = "Wildcat! Local 0220"
Const MaxCallers = 23, MaxMinOnline = 30, MinUpload = 45000
Const FuzzyBlock = Chr(176)
Dim YesSir As String*1
Print "Do you want to see our stats? (YN) ";
```

```
Input YesSir
If (Asc(YesSir) >= 97) And (Asc(YesSir) <= 122) Then
   Print String(39, FuzzyBlock)
   Print "This board's name is ";LocalBBSName
   Print "----------------------------------------"
   Print "The maximum lines available is ";MaxCallers; "."
   Print "Each caller has ";MaxMinOnline; " minutes on-line."
   Print "There is a mandatory minimum upload of "; MinUpload; "
bytes."
End If
```

COPYFILE Function

Copy a file to another directory or drive.

Syntax

CopyFile(*SourceFile*, *TargetFile*) {As True/False}

Remarks

The **CopyFile** function does just as the name describes, it copies a file from one location to another. It will issue no warning, prior to over-writing a file with an identical name, if one exists in the target directory. The function will return either **True** or **False** upon the success of the operation. Both *SourceFile* and *TargetFile* must include the file name as a path-only in *TargetName* will be rejected.

See also

Name

Example

```
Rem Copy a file across drives

If CopyFile("C:WCLIST.OUT", "D:\BAK\DATA\WCLIST.BAK") Then
   Print "File copied"
Else
   Print "Copy failed"
   Beep
   WaitEnter
End If
```

COS Function

Return the cosine of an angular numeric expression.

Syntax

Cos(*AngleVal*) {As Real}

Remarks

The argument *AngleVal* is an angle expressed in radians. Radians can be converted to degrees by multiplying by 180/pi where pi is approximately 3.14159.

See also

Atn, Sin, Tan

Example

```
Rem Print the Cosine of 1 Radian

Print Cos(1)
```

CSRLIN Function

Return the screen line number of the cursor position.

Syntax

CsrLin {As Byte}

Remarks

This function will return the screen line number where the cursor is currently positioned.

See also

Locate, Pos

Example

```
Rem Store the cursor positions and step through the database

Dim NameRow As Integer, NameCol As Integer
Dim StateRow As Integer, StateCol As Integer
Dim uRec As UserRecord, uName As String
Print "User's name is    ";
NameRow = CsrLin
NameCol = Pos
Print "From the state of ";
StateRow = CsrLin
StateCol = Pos
Print
Rem Null string for user name will start at beginning
Print "Starting user name: ";
Input uName
If GetUser(uRec, uName) Then
   Do
```

```
        Locate NameRow, NameCol
        Print uRec.Name
        Locate StateRow, StateCol
        Print uRec.State
        If Not(InputYesNo("More...")) Then Exit Do
    Loop Until Not(GetNextUser(uRec))
End If
```

CURRENTDATE Statement

Return the current date.

Syntax

CurrentDate *DateVar*

Remarks

This statement will return the current date to a variable *DateVar* of the internal data type, **Date**. To use this returned value the **FormatDate** function is called with a string representation of the desired format.

See also

CurrentDateTime, CurrentTime, FormatDate, FormatTime

Example

```
Rem Display today's date and format the display

Const DateMask = "Mm/dd/yyyy"
Dim Today As Date
CurrentDate(Today)
Print "Date: "; FormatDate(Today, DateMask)
```

CURRENTDATETIME Statement

Return the current date and time.

Syntax

CurrentDateTime *DateTimeVar*

Remarks

This statement will return the current date and time to a variable *DateTimeVar* of the internal data type, **DateTime**. To use this returned value the **FormatDate** and **FormatTime** functions are called with a string representation of the desired format.

See also

CurrentDate, CurrentTime, FormatDate, FormatTime

Example

```
Rem Get the current date and time and format the result

Const DateMask = "DD/mm/yyyy"
Dim TodayNow As DateTime
CurrentDateTime(TodayNow)
Print "Date: "; FormatDate(TodayNow.D, DateMask);
Print " Time: "; FormatTime(TodayNow.T, "HH:MM")
```

CURRENTTIME Statement

Return the current time.

Syntax

CurrentTime *TimeVar*

Remarks

This statement will return the current time to a variable *TimeVar* of the internal data type, **Time**. To use this returned value the **FormatTime** function is called with a string representation of the desired string format.

See also

CurrentDate, CurrentDateTime, FormatDate, FormatTime

Example

```
Rem Get the current time and format the output

Const TimeMask = "Hh:MM:Ss"
Dim TimeNow As Time
CurrentTime(TimeNow)
Print "Time: "; FormatTime(TimeNow, TimeMask)
```

DECLARE Statement

Forward declare a subroutine or function.

Syntax

Declare Function *Name* [(*Arg* **As Type** [, *Arg* **As Type**...])] **As Type**

Declare Sub *Name* [(*Arg* **As Type** [, *Arg* **As Type**...])]

Remarks

The **Declare** statement is to inform the compiler that you intend to call a subroutine or function in your program that it has not yet encountered but it will do so by the end of the compiling process. Using **Declare** allows you to put the bulk of your program at the start of the text file so you don't have to scroll through pages of subroutines and functions to get to it.

The *Name* is the name of the subroutine or function which you make up to describe just what it does. The *Arg* or *Arg* list **Declare**d must exactly match that of the complete definition of the subroutine or function later in your program source code.

The **Declare** statement can be optional unless a circular reference has to be made from within an existing subroutine or function. For example if one function needs to call another function following it but it also has a call to the previous function then a **Declare** statement becomes mandatory. In such a case the first function, by having no knowledge of the second's existence would force an error at compile time.

On occasions it can be useful to have a subroutine or function that will take a varying number of parameters in the argument list. The *wcCODE* language can handle this situation very elegantly, as long as two conditions are met when defining the Sub or Function.

The first condition is that the routine must be pre-declared using the "Declare" statement. The second condition applies to the parameters that are to be optional. These optional parameters must have their default value assigned in the declaration portion, but not in the definition of the

actual routine. For an example of this process, see Chapter 2, *wcCODE programming*, in the section headed, Subs and Functions.

See also

Call, Function, Sub

Example

```
Rem A simple forward declaration function

Declare Function CapName(tStr As String) As String
Dim FName As String*30, LName As String*30
Dim Prompt(2) As String*5, Ctr As Integer
Prompt(1) = "first" : Prompt(2) = "last"
Ctr = 0
Do
   Ctr = Ctr + 1
   Print "Enter the "; Prompt(Ctr); " name of your favorite actor ";
   Select Case Ctr
   Case 1
      Input FName
   Case 2
      Input LName
   End Select
Loop Until Ctr = 2
Print "So "; CapName((FName)); " "; CapName((LName)); " eh?"

Function CapName(tStr As String) As String
   CapName = UCase(Left(tStr, 1)) + \
      LCase(Mid(tStr, 2, Len(tStr) - 1))
End Function
```

DEFCOLOR Function

Return the default color.

Syntax

DefColor {As String}

Remarks

The **DefColor** function will return the default color in the standard *Wildcat!* format for colors of @FB@ (called At Codes). Because *Wildcat!* takes the @Codes and converts them into a screen color attribute it is not a straight forward exercise for you to display just what **DefColor** has in fact returned.

You cannot print these @Codes without them being interpreted, so the easiest way is to drop the leading @ and then you will be able to print the remainder of the string returned by **DefColor**. This can be achieved by printing the right-most 3 characters using the **Right** statement.

DefColor is best used to reset the screen colors back to the default color after your program has made its Technicolor dream coat messages. Just calling **DefColor** on its own won't do it though, you will need to call it in a **Print** statement for it to change the colors.

See also

Cls, GetColor

Example

```
Rem Display colors and reset with DefColor

Const RedYellow = "@E4@", BlueWhite = "@F1@"
Print RedYellow + "Fried eggs and Ketchup"
Print BlueWhite;
Print "Sailors ahoy!"
Print DefColor; "Finally black to sanity."
```

DEL Statement

Delete a file from disk.

Syntax

Del *FileName*

Remarks

The name *FileName* is the file to delete. This statement is to delete single files only. DOS wildcards * and ? are not supported.

To delete groups of files use **Del** in conjunction with **FindFirst** and **FindNext** procedures where the DOS wildcard characters *are* supported and the full file name can be passed to the **Del** statement.

You cannot delete a file that is already open by your *wcCODE* program or hopefully any other program but that responsibility must fall to you, your program and the operating system to monitor. DOS may let you delete an open file which can often lead to disastrous results. It is therefore your responsibility to ensure that all files to be deleted are under your program's control and have been closed.

Using the **Close** command without any parameters will close *all* open files in use by your program.

The **Del** and **Kill** commands are identical in their operation and can be freely interchanged. The reason we include both is to maintain consistency with the original BASIC language on which *wcCODE* is based. This duplication is to assist those programmers who prefer to use keywords or commands they are more familiar with rather than dictate our own opinions on what *we* think they should use.

See also

Close, **FindFirst**, **FindNext**, **Kill**, **RmDir**

Example

```
Rem Attempt to delete the Test.DAT file

Const FName = "Test.DAT"
If Exists(FName) Then   : Rem First check to see if it is there
   Del(FName)
Else
   Print "Could not find "; FName
End If
```

DELAY Statement

Suspend program execution for a defined period of time.

Syntax

Delay *Time*

Remarks

The *Time* value is the duration to suspend the execution of the program in seconds. For a duration less than a full second, fractional decimal values may be used.

See also

Pause, WaitEnter

Example

```
Rem Reset a Hayes type MODEM, then dial a number

Delay(3)          : Rem Get the attention of the Modem first
Send "+++"
Delay(3)
Send "ATZ" + Chr(13) : Rem Send a carriage return too
Delay(3.50)
Send "ATDT *70" + Chr(13) : Rem Turn Call-waiting on or off
```

DELETEMESSAGE Statement

Delete a message

Syntax

DeleteMessage

Remarks

The **DeleteMessage** statement will present the user with the standard [K]..Kill message response as though the option had been selected from a menu.

The DeleteMessage command presents the user with the standard prompt for selecting the message number to delete or the option of aborting the process.

Note!

Do not confuse this use of the Message menu [K]..Kill with the *wcCODE* program statement **Kill** for deleting a file.

Example

```
Rem Clean up the message area

Print "About to begin deleting messages."
Do While InputYesNo("Kill more? ")
   DeleteMessage
Loop
```

DIM Statement

Declare variables of any data type.

Syntax

Dim *Name* As *Type* [, *Name* As *Type*...]

Dim *Name*([*Lo* To] *Hi*) As *Type* [, *Name*([*Lo* To] *Hi*)] *As Type*...]

Remarks

All variables *must* be declared with the **Dim** statement before they can be used. Each variable must have an associated data type assigned to it by using the **As *Type*** declaration. The data type must be an internal data type, a pre-defined *Wildcat!* data type or a user-defined data type.

If variables are declared on the same text line they must be separated by commas and different data types may be freely mixed on the one line. If separate lines are preferred then the **Dim** statement must be used at the beginning of each text line.

The second construct is for declaring arrays. Arrays are accessed by their subscripts, the value between the parentheses. When declaring arrays with the **Dim** statement the '*Lo* **To**' construct can be omitted as all arrays will default to zero based if no *Lo* bound is declared.

The only time a '*Lo* **To**' statement would be useful is when dealing with a small range of array elements with unusually high subscripts. If for some reason, you wanted to reference a group of array elements starting with a *Lo* of 1200 and a *Hi* of 1230 it would waste a whole lot of memory by not declaring the lower bounds. The elements 0 through 1199 would never be used in such an example.

See also

Const, Static

Example

```
Rem Typically declare a few variables

Dim ByteA As Byte : Dim TestStrA As String
Dim ByteB As Byte, TestStrB As String*128
Dim IntA As Integer
Dim PayDayTotal as Real
Dim ZeroBaseArray(56) As Long          : Rem 0 to 56 subscripts
Dim MyArray(15 To 109) As String*64     : Rem 15 to 109 subscripts
For ByteA = 0 To 56
   ZeroBaseArray(ByteA) = ByteA * 6
Next
For ByteA = 0 To 56
   Print ByteA; " value of array = "; ZeroBaseArray(ByteA)
Next
```

DISPLAYFILE Function

Display the contents of a text file.

Syntax

DisplayFile(*FileName*) {As True/False}

Remarks

The **DisplayFile** command will simply list out the contents of the file named *FileName*. The FileName must exist in the current conference directory with an extension of .BBS, .SCR or .RIP. The extension need not be specified when using this command as the appropriate file for the caller's configuration will be used.

Being a Boolean type **Function** the **DisplayFile** will return a **True/False** result on the success or failure of the function.

See also

DisplayTextFile, SearchFiles

Example

```
Rem Greet the caller

Dim Today As Date
Const DateMask = "dd/mm/yy"
CurrentDate(Today)
If User.BirthDate.Number = Today.Number Then
   If Not(DisplayFile("Birthday")) Then
      Print "Birthday file was not found!";
   End If : Rem DisplayFile failed
End If : Rem Birthday today
```

DISPLAYGROUPFILES Statement

Display group files.

Syntax

DisplayGroupFiles

Remarks

This command checks the GROUP###.LST files for the caller's name. If the name is found then the appropriate files, if any exist will be displayed.

The DisplayGroupFiles command is intended for use in a logon program.

See also

CheckMail, CheckMailAtLogon, CheckNewsLetter, CheckBulletins

Example

```
Rem A very basic logon program

QuoteOfTheDay
CheckNewsLetter
CheckBulletins
CheckMailAtLogon
DisplayGroupFiles
```

DISPLAYHELP Statement

Display help file contents.

Syntax

DisplayHelp *FileName*

Remarks

This command will display the contents of the *FileName* help file, if it exists in the HELP directory.

Example

```
Rem Display the help file to a caller

Dim FName As String
FName = "\WILDCAT\HELP\FaxHelp.BBS"
If Exists(FName) Then
   DisplayHelp FName
Else
   Print "You failed to find help on the Fax feature"
   WaitEnter
End If
```

DISPLAYPROMPT Statement

Display a prompt from the prompt file.

Syntax

DisplayPrompt *Num*

Remarks

The **DisplayPrompt** command will display the prompt associated with the *Num* value. This can be a variable or it can be an actual number.

For the numbering of prompts use *wcPROMT* and load WILDCAT.PRM. The number associated with the prompt will be displayed in the top of the screen as each prompt is displayed.

The advantage of using **DisplayPrompt** is that different language versions of your program are automatically converted when the Language option is changed in *MAKEWILD*. If the prompt you want is not in the *wcPROMT* list then you could also consider your own prompt file and access it using the **Seek** command.

Note!

For another example of using **DisplayPrompt** see the code fragment supplied with **AddMessage**.

Example

```
Rem Add multilingual support by using Wildcat! prompts

Dim tRec As UserRecord
Print "New name to add to the database: ";
Input tRec.Name
If AddUser(tRec) Then
   Print "New name added"
Else
   DisplayPrompt 67 : Rem Run program to see 67?
End If
```

DISPLAYSTOPPED Function

Indicate if the caller selected Stop from the Pause command.

Syntax

DisplayStopped {As True/False}

Remarks

This function will indicate whether the caller pressed [S]top in response to the **Pause** command or the SPACE bar was pressed during a non-stop display of text to the screen.

See also

MorePrompt

Example

```
Rem Assume the file is a long text file
Const MaxCtr = 200
Dim Ctr As Integer, KeepGoing As Integer
Dim tStr As String
Open "LongText.TXT" For Input As #1 : Rem Assume it exists
KeepGoing = InputYesNo("Display large lumps at a time? ")
If KeepGoing Then MorePrompt Off
While Not(EOF(1))
   Ctr = Ctr + 1
   Input #1, tStr : Print DefColor; tStr
      Rem Check every so often if large lumps displayed
   If (KeepGoing) And (Ctr > MaxCtr) Then
      Ctr = 0
      Pause
      If DisplayStopped Then GoTo HadEnough
   End If
Wend
HadEnough:
Close #1
```

DISPLAYTEXTFILE Function

Display the contents of a text file.

Syntax

DisplayTextFile(*FileName*) {As True/False}

Remarks

This command as the name implies will display the contents of the *FileName* text file. A full path can be used with the file name to locate the file. The function returns **True** or **False** on the success of the operation.

Example

```
Rem Greet the caller


Dim Today As Date
Const DateMask = "mm/dd/yy"
Const FName = "\WILDCAT\LANGUAGE\Birthday.TXT"
CurrentDate(Today)
If User.BirthDate.Number = Today.Number Then
   if Exists(FName) Then
      DisplayTextFile(FName)
   Else
      Print "Birthday.TXT was not found"
   End If
End If : Rem Birthday today
```

DO Statement

Repeatedly execute a block of statements.

Syntax

Do [{**While** | **Until**} *Condition*]

Statements...

Loop

Do

Statements...

Loop [{**While** | **Until**} *Condition*]

Remarks

The **Do** statement is never used without a partnering **Loop** statement therefore the example here is exactly the same as that for **Loop**.

The first construct above tests for the specified *Condition* being met at the beginning of the loop and exits as soon as the condition is satisfied. If the condition has already been met when this construct is encountered then the statements contained within the **Do**...**Loop** *will not* be performed.

The second construct tests for the specified *Condition* at the end of the loop and continues until the condition is met. If the *Condition* has already been met when this construct is encountered the statements contained within the **Do**...**Loop** *will* be performed once before exiting.

The *Condition* is any logical expression that evaluates to either **True** (-1) or **False** (0).

The keyword **While** repeats the loop while the *Condition remains* true whereas the keyword **Until** repeats the loop until the *Condition becomes* true.

The *Statements* are valid *wcCODE* program statements that will be performed for each iteration. A statement block is optional as a loop may

be empty while waiting for an external event such as a key-press or an incoming character.

See also

Exit Do, For...Next, While...Wend

Example

```
Rem A basic example of a Do ... Loop

Dim ByteA As Integer

Const WaitingStr = " I'm still here and working hard"

ByteA = 0

Do

    Print ByteA; WaitingStr

    ByteA = ByteA + 1

Loop Until ByteA > 10

Rem ByteA now equals eleven

Do While ByteA < 10

    Print "This will never be seen"

    ByteA = ByteA - 1

Loop
```

DOOR Statement

Run a Door program.

Syntax

Door *Num*

Remarks

This command will run the **Door** program *Num* if it exists. If no **Door** is found then an error message is displayed.

See also

DoorMenu

Example

```
Rem As part of a menu selection this could run a door

Door 1
```

DOORMENU Statement

Display the door menu.

Syntax

DoorMenu

Remarks

This command will display the Door menu if one exists. If there are no Doors available or the user does not have access to them then an error message is displayed. This command has exactly the same response as selecting the menu item, "Door menu."

See also

Door

Example

```
Rem A very basic use of the DoorMenu command

DoorMenu
```

DOSVERSION Function

Return the current DOS version.

Syntax

DosVersion {As String}

Remarks

This function will return the DOS version as a string in the format of major version number, period then minor version number. For example, DOS version 6.0 would be reported as "6.0".

See also

Version

Example

```
Rem Determine if the DOS version is suitable

Function DosGreaterThan(LowestVer As Byte) As Integer
   Dim DVStr As String, DV As Integer
   DosGreaterThan = False
   DVStr = DosVersion
   DV = Val(Left(DVStr,Instr(DVStr,".") - 1))
   If DV < LowestVer Then
      Print "wcCODE requires DOS "; LowestVer; \
            ".xx or better. Your version is "; DVStr
   Else
      DosGreaterThan = True
   End If
End Function
If DosGreaterThan(5) Then : Rem Try 9 here too
   Print "This looks good already!"
Else
   Print "program ending"
   End
End If
```

DOWNLOAD Statement

Download a file.

Syntax

DownLoad

Remarks

This command has exactly the same response to the caller as if they had selected the [D]..Download a file menu option.

See also

SysopDownload, Upload

Example

```
Rem Allow your program to control downloads

Dim tStr As String*1
Do
   Print "Press [D]..Download a file   [Q]..Quit back to menu ";
   Input tStr
   Select Case UCase(tStr)
      Case "D"
         DownLoad
      Case "Q"
         End
   End Select
Loop : Rem No need for a condition as "Q" will end the loop
```

EDITMARKLIST Statement

Edit the list of files marked for downloading.

Syntax

EditMarkList

Remarks

When the **EditMarkList** is called the user is presented with the same response as if they had selected the [E]..Edit marked list menu option.

Example

```
Rem Process the marked list of files

If InputYesNo("Do you need to edit the list? ") Then
   EditMarkList
End If
```

ELSE & ELSEIF Statement

Provide alternatives to an If-condition not being met.

Syntax

If *Condition* Then

Statements...

[ElseIf *PartCondition* Then

Statements...]

Else...

Statements...

End If

Remarks

The **Else** statement is used within an **If** statement to provide an alternative to the primary *Condition* being met.

The **ElseIf** (Note *no* space between **Else** and **If**) option allows for testing additional conditions.

In the following example the first branching where all three conditions are tested would give unlimited access to whatever area the user was in. The **ElseIf** statement then allows another condition based on the first failing but at least the first two fitting the requirements for browsing rights. If all else fails then the user should not be in here and therefore gets no access at all.

The **Else** statement can also be used within the **Select Case** construct as an alternative **Case Else** condition. **Case Else** acts like a 'catch all' process for all items not meeting any of the declared **Select Case** list of conditions.

See also

If, Select case, Then

Example

```
Rem Else and ElseIf used here to control partial If

Const MaxID = "SYSOP", MaxAcc = 20, SysPw = "TESTING"
Dim uID As String, uAcc As Integer, uPw As String
Print "Enter your ID please "; : Input uID
uID = UCase(uID)
Print "Enter your access level please "; : Input uAcc
Print "Enter your access level "; uAcc; " password ";
Input uPw : uPw = UCase(uPw)
If (uID = MaxID) And (uAcc = MaxAcc) And (uPw = SysPw) Then
   Print "Access to all areas is valid for unlimited time."
ElseIf (uID = MaxID) And (uAcc = MaxAcc) Then
   Print "Access is limited to Browse for 6 minutes."
Else
   Print "Access is denied with those access levels."
   Beep
End If
```

ENCODEPASSWORD Function

Encode a password for the user database.

Syntax

EncodePassword(*String*) {As String}

Remarks

When adding a new user to the database this function will encode the password *String* in the same manner as the *Wildcat!* system.

Note!

When adding a user with the **AddUser** function the password is not automatically encoded. You must use **EncodePassword** before calling **AddUser** with the user record.

See also

AddUser

Example

```
Rem You must encode a password before updating

Dim tStr1 As String, tStr2 As String, uRec As UserRecord
Do
  Print "Password ";
  Input tStr1
  tStr1 = EncodePassword(tStr1)
  Print "Verify ";
  Input tStr2
  tStr2 = EncodePassword(tStr2)
  If tStr1 <> tStr2 Then
    Print "Passwords did not verify. Please re-do"
    Beep
```

```
  End If
Loop Until tStr1 = tStr2
uRec.Password = tStr1
AddUser uRec : Rem Assume uRec was filled in before password
```

END Statement

End a program.

Syntax

End

Remarks

The **End** statement immediately terminates the currently executing program and returns to normal *Wildcat!* operation. Any currently open files are closed before the program ends.

The **End** statement may be used anywhere within a program, subroutine or function to immediately terminate the process.

See also

Goodbye, HangUp, Stop

Example

```
Rem Force the termination of the wcCODE program

Dim Ctr As Integer, PwStr As String*5
For Ctr = 1 To 3
   Print "Enter your password "; Ctr; ": ";
   Input PwStr
   If UCase(PwStr) <> "SYSOP" Then
      End : Rem Terminate the process right now!
   End If
Next Ctr
Print "Ok so far so good"
```

ENTERMESSAGE Statement

Enter a message to the message database.

Syntax

EnterMessage

Remarks

The EnterMessage command is exactly the same as if the user had selected the [E]..Enter message menu option.

See also

AddMessage

Example

```
Rem Your program can process messages

Print "You are about to leave the RIP conference."
If InputYesNo("Add a message before leaving? ") Then
   EnterMessage
End If
```

ENVIRON Function

Return the contents of a DOS environment variable.

Syntax

Environ(*EnvName*) {As String}

Remarks

The *EnvName* is the name of the environment variable whose data you wish to retrieve. The *EnvName* parameter should be in upper case. If the specified environment variable is not found, an empty string will be returned.

Example

```
Rem Display the DOS path string

Dim EnvStr As String
Print "Options are: COMSPEC, PATH, PROMPT ";
Input EnvStr
EnvStr = UCase(EnvStr)
Select Case EnvStr
   Case "COMSPEC", "PATH", "PROMPT"
      Print "The "; EnvStr; "environment variable = ";
      Print Environ(EnvStr)
   Case Else
      Print "Invalid environment variable "; EnvStr
End Select
```

EOF Function

Determine if the end of a file has been reached.

Syntax

EOF(*FileNum*) {As True/False}

Remarks

This function will return **True** during a read of the file using **Input** or **Get** if the end of the file specified by the *FileNum* parameter has been reached.

See also

Loc, LOF, Seek

Example

```
Rem Print the entire contents of a text file

Dim InputStr As String
Open "Test.DAT" For Input As #1 : Rem Assume it exists
Do While Not EOF(1)
    Input #1, InputStr
    Print InputStr
Loop
Close #1
```

EQV Operator

Perform a logical Equivalence operation between operands.

Syntax

Op1 Eqv *Op2*

Remarks

Essentially **Eqv** compares the values on either side for equivalence. This means that if both are **True** or both are **False** then they are considered equivalent.

The *Op1* and *Op2* operands are logical expressions or numeric values of any data type with the *exception* of the **String** type.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

And, Imp, Not, Or, XOr

Example

```
Rem A demonstration of the Boolean operator

Dim i As Integer, j As Integer, k As Integer
i = 10 : j = 8 : k = 6
If i > j Eqv j > k Then : Rem Same as below!
   Print "Yup, i is greater than j and j is also greater than k"
End If
i = 6 : j = 8 : k = 10
If i > j Eqv j > k Then : Rem Same as above!
   Print "But, i is LESS than j and j is also LESS than k"
End If
```

ERROR Statement

Force a defined runtime error.

Syntax

Error *ErrValue*

Remarks

When called at run-time the error defined by *ErrValue* is forced. If there is a **Catch** handler available for *ErrValue* then it will be executed. Otherwise the program will be terminated automatically under the conditions outlined in the **Catch** statement for an undefined error.

This is basically a debugging aid in being able to force a particular type of error to see how well or how badly your program handles the situation. You can also use **Error** to define your own error values which can be handled with the **Catch** statement just *wcCODE*'s internal error codes can.

If you do define your own error codes, then it is recommended that you select values higher than 20. This should prevent any conflict with *wcCODE*'s internal use of numbers in this range.

However **Error** can also be used within a **Select Case** statement or in fact anywhere in your program to force an error to exit a subroutine or function if you are not happy with a user's response. A preferred and more elegant method to handle an errant user would be to print a message and simply end the program or immediately exit the subroutine or function.

See also

Catch

Example

```
Rem Catch a file error

Sub UserCode(AllOk As Byte)
   Dim tStr As String, Ctr As Integer
   Const Err_NoGoJoe = 21
   Ctr = 0
   AllOk = 99 : Rem Get ready for a good exit
   While (tStr = "" ) And (Ctr < 4)
      Ctr = Ctr + 1
      Print "Enter your user code ";
      Input tStr
   Wend
   If (Ctr > 3) And (UCase(tStr) <> "SUCCESS") Then
      Error(Err_NoGoJoe)
   End If
   Catch Err_NoGoJoe
      Print "Bye bye. It's back to the menu for you!"
      Delay(3)
      AllOk = 0
End Sub

Dim Ok As Byte
UserCode(Ok)
If (Ok < 1) Then
   End : Rem Dump the user back to the menu
Else
   Print "Quite acceptable!"
End If
```

EXISTS Function

Determine whether a file exists on disk.

Syntax

Exists(*FileName*) {As True/False}

Remarks

This function will return **True** if the specified file *FileName* exists on the disk. A full path must be specified if other than the current directory is queried for the file's existence. DOS wildcard characters '?' and '*' are not allowed.

See also

FindFirst, FindNext

Example

```
Rem Check for a file to prevent an opening error

Function OpenFile(FName As String) As Integer
    OpenFile = True : Rem Assume the best
    If Exists(FName) Then
        Open FName For Input As #1
    Else
        OpenFile = False
    End If
    Catch Err_FileOpen
        OpenFile = False
End Function        : Rem Assume file exists
Dim tStr As String : Const tFName = "TestData.TXT"
If OpenFile(tFName) Then
    While Not(EOF(1))
```

```
        Input #1, tStr
        Print tStr
    Wend
Else
    Print "Error trying to open the file, wait. ";
    Delay(3)
End If
```

EXIT Statement

Exit immediately from a loop, subroutine, or procedure.

Syntax

Exit Do

Exit For

Exit Function

Exit Sub

Remarks

One of the appropriate keywords from the four above *must* be included in the **Exit** statement.

Use **Exit Do** when in a **Do**...**Loop** structure.

Use **Exit For** when in a **For**...**Next** structure.

Use **Exit Function** to prematurely exit from a **Function**

Use **Exit Sub** to prematurely exit from a **Sub**routine.

If you are exiting a function with **Exit Function** and no value has been assigned to the function result, the function will return zero or an empty string, depending on the data type of the function. Ideally a default function result should be established very early on in the function code in anticipation of an early exit.

See also

Do...Loop, For...Next, Function, Sub

Example

```
Rem Get out of a function early

Function IncToMax(ByteA As Byte, MaxByte As Byte) As Integer
   IncToMax = False : Rem Just in case we have to quit early
   If (ByteA + 1) > MaxByte Then
      Exit Function
   Else
      ByteA = ByteA + 1
      IncToMax = True
   End If
End Function

Dim Ctr As Byte, Row As Integer, Col As Integer
Dim Key As String
Const MaxCtr = 20
Ctr = 0
Row = CsrLin : Col = Pos
Do While IncToMax(Ctr,MaxCtr)
   Locate Row, Col
   Print "You have until the counter equals "; MaxCtr;
   Print " Now at "; Ctr
   Delay 0.5
   Key = Inkey : Rem Any keystrokes pending?
   If Key > "" Then Exit Do
Loop
```

EXP Function

Return a value raised to a specified power.

Syntax

Exp(*Val*) {As Real}

Remarks

The **Exp** function will return e raised to a specified power, where e represents the base of natural logarithms.

See also

Log, Exp

Example

```
Rem Compute the square of the natural Logarithm base (e)

Print Exp(2)
```

FASTLOGON Function

Return a result on the caller's logon method.

Syntax

FastLogon {As True/False}

Remarks

The **FastLogon** command will determine whether the caller used the fast logon procedure or went through the normal prompts and answers sequence. The fast logon feature of *Wildcat!* is activated by the caller using the asterisk (*) or exclamation point (!) preceding their name during the logon process.

The **FastLogon** function could be used by your program to provide other short cuts for the experienced user.

Example

```
Rem Determine the caller's logon method

Dim AllowShortCuts As Integer
AllowShortCuts = False
If FastLogon Then
   AllowShortCuts = True
End If
```

FILEINFO Statement

Display the extended description for a file.

Syntax

FileInfo

Remarks

This command has exactly the same response to the user having selected [I]..Info on a file when using the file system. It is used to display any extended description in the files database for that file.

See also

GetFileInfo

Example

```
Rem Having selected a file, display extended description

FileInfo
```

FILESTATS Statement

Display the statistics on file transfers.

Syntax

FileStats

Remarks

This command has exactly the same response to the user having selected [S]..Stats when using the file system.

Example

```
Rem Display the file statistics

FileStats
```

FINDFIRST Function

Find the first file to match a file name specification.

Syntax

FindFirst(*FileMask*, *Attr*, *SrchRec*) {As Integer}

Remarks

This function is used to locate the first file name that matches the specified *FileMask*. Once it has done so it will return the standard DOS information for the file. The *FileMask* is the file specification to search for which can include the DOS wildcard characters * and ? and a complete DOS path.

Attr is the file attribute to look for. Generally this will be to search for "normal files" so use 0 (zero) for this parameter.

The *SrchRec* is a variable of type **SearchRec** that will hold the file information for the file found. **SearchRec** is an internal variable structure which will contain the file name, the time and date in a compressed format, the size of the file and its attribute byte.

This function is normally used in conjunction with the **FindNext** function to gather a list of files to perform other operations on such as **Del** or **Kill.**

FindFirst will return zero if at least one file is found that matches the specification defined in *FileMask* otherwise it will return a non-zero result indicating the operation failed. **FindFirst** as with **FindNext** will actually return the DOS error value for a value other than zero. You could if you wish, trap this specific error and handle it within your program. For a list of DOS *file* error messages refer to the appendix at the rear of this manual.

Note!

For more information on the structure of the **SearchRec** type, refer to the appendix in this manual.

For more information on the packed **Time** and **Date** information returned in **SearchRec** refer to the appendix in this manual.

Valid file attributes

Dec	Hex	Description
0	0	Normal archive file
1	1	Read only file
2	2	Hidden file
4	4	System file
8	8	Volume label
16	10	Directory name

See also

Exists, FindNext

Example

```
Rem Display all of the available files with a .TXT extension

Dim SRec As SearchRec, FileFound As Integer
Const FileMask = "D:\WILDCAT\MSG\TXTDATA\*.TXT"
FileFound = FindFirst(FileMask,0,SRec)
Do While FileFound = 0
   Print SRec.Name; " ";
   Print SRec.Size; "  "
   FileFound = FindNext(SRec)
Loop
```

FINDNEXT Function

Finds the next file in the set defined by the FindFirst function.

Syntax

FindNext(*SRec*) {As Integer}

Remarks

This function must always be used in conjunction with the **FindFirst** function to search for a list of files that match the *FileMask* passed to **FindFirst**.

FindNext will return zero if at least one file is found that matches the specification defined in the original **FindFirst** *FileMask* otherwise it will return a non-zero result indicating the operation failed. **FindNext** as with **FindFirst** will actually return the DOS error value for a value other than zero. You could if you wish, trap this specific error and handle it within your program. For a list of DOS *file* error messages refer to the appendix at the rear of this manual.

Note!

If you are using **FindFirst** and **FindNext** to delete a group of files before removing a temporary directory then you must remember to add the path name to the file name when calling the **Del** command as **SearchRec.Name** contains only the actual file name.

See also

Exists, FindFirst, Del, Kill

Example

```
Rem Delete unwanted .MSG files

Function YesNo(Prompt As String) As Integer
Dim Key As String*1
   Print Prompt;
   Do
      Key = Inkey
   Loop until Key <> ""
   YesNo = UCase(Key) = "Y"
End Function

Dim SRec As SearchRec, FileFound As Integer
FileFound = FindFirst("*.MSG",0,SRec)
Do While FileFound = 0
   Print SRec.Name;
   If YesNo(" Delete (YN) ") Then
      Del SRec.Name
   End If
   Print
   FileFound = FindNext(SRec)
Loop
```

FIX Function

Return the integer portion of a Real value.

Syntax

Fix(*Number*) {As Long}

Remarks

Fix and Int operate the same for positive values, however with negative values their operation is slightly different. Fix returns the next-higher integer with negative values, while Int returns the next lower integer with negative values.

The example code will return 3 and -4 for Fix and 3 and -5 for Int.

See also

Int

Example

```
Rem Fix and Int examples

Print "Fix(3.7) = "; Fix(3.7)
Print "Int(3.7) = "; Int(3.7)
Print "Fix(-4.9) = "; Fix(-4.9)
Print "Int(-4.9) = "; Int(-4.9)
```

FOR Statement

Execute a sequence of statements a specified number of times.

Syntax

For *Index* = *StartVal* To *EndVal* [Step *StepVal*]

Statements...

Next [*Index*]

Remarks

The *Index* is a numeric variable of type **Byte**, **Integer**, **Long**, or **Real** which is incremented each time the loop is performed. The *StartVal* value is the starting point or counter of the loop. The *Index* is set to this value at the start of the **For**...**Next** sequence. The *EndVal* value is that maximum value for the *Index* to reach for the **For**...**Next** process to terminate.

The *Step* is an optional designator that defines the amount by which the counter is incremented after each pass in a **For**...**Next** sequence. When the *Step* value is omitted the increment defaults to one. The *Step* value can also be a negative number to allow a decrement downwards.

The *Index* following the **Next** statement is optional but for programming clarity it is a good idea to include it. This is especially true when you have several nested **For**...**Next** loops all ending with a series of **Next Next Next** statements without the Index designators. Do yourself a favor and prepare for easy debugging when needed.

See also

Do...Loop, Exit For, While...Wend

Example

```
Rem Display a count-down timer

Dim Ctr As Integer, Row As Integer, Col As Integer
Dim Dot As Integer
Print "Waiting for keyboard response : ";
Row = CsrLin : Col = Pos
For Ctr = 30 To 1 Step -1
   Locate Row, Col
   Print Ctr; " "; : Rem Stomp on extra digit when we get below 10
   For Dot = 1 To 10
      Print ".";
      Delay 0.1
      If InKey > "" Then
         Dot = 99
         Exit For : Rem Exit the Dot's For-Next
      End If
   Next Dot
   If Dot = 99 Then Exit For : Rem Exit the Ctr's For-Next
   Locate Row, Col
Next Ctr
If (Dot < 99) Or (Ctr = 0) Then Print "Timed out"
```

FORMATDATE Function

Format the Date data type.

Syntax

FormatDate(*Date*, *Mask***)** {As String}

Remarks

This function will take the numeric data in the **Date** or **DateTime.D** variable and place the appropriate components in-place of the *Mask* characters.

The *Mask* is a string of characters representing the components of the date to display. In the **FormatDate** function the letters 'd', 'm', 'y' represent days, months and years.

Only the mask characters you require need be used. For instance if you only want to display the month and the year then the *Mask* of "mm/yy" will do the trick.

Using uppercase letters for the mask characters will display a blank space in place of the zero normally displayed if lowercase letters were used. e.g. "dd/mm/yy" returns "23/07/94" where "dd/DD/YY" returns "23/ 7/94", your choice.

See also

CurrentDate, CurrentDateTime, CurrentTime, FormatTime, .Number

Example

```
Rem Different demos of the Format feature

Dim tDate As Date, tTime As Time, tDateTime As DateTime
Dim tStr As String
CurrentTime(tTime)
CurrentDate(tDate)
CurrentDateTime(tDateTime)
Print FormatTime(tTime, "hh:mm:ss"),
Print FormatDate(tDate, "dd/mm/yyyy")
Print FormatTime(tDateTime.T, "hh:mm"),
Print FormatDate(tDateTime.D, "dd/mm/yy")
tStr = "The time is now : " + \
      FormatTime(tDateTime.T, "hh:MM:SS")
Print tStr
```

FORMATTIME Function

Format the Time data type.

Syntax

FormatTime(*Time*, *Mask***)** {As String}

Remarks

This function will take the numeric data in the **Time** or **DateTime.T** variable and place the appropriate components in-place of the *Mask* characters.

The *Mask* is a string of characters representing the components of the date or time to display. In the **FormatTime** function, the letters 'h', 'm', 's' represent hours, minutes and seconds. The time can also be displayed with "am", "pm", "a" or "p" by using the letters "te" and "t" respectively, appended to the mask characters. The letters "te" may seem a little obscure for the purpose, but they are the internal representation for am/pm formats.

When using the am/pm function the "te" or the "t" must be in lower case letters, otherwise they will be treated as literals and printed as such.

Only the mask characters you require need be used. For instance if you only want to display the minutes then the *Mask* of "mm" will do the trick.

Using uppercase letters for the mask characters will display a blank space in place of the zero normally displayed if lowercase letters were used. e.g. "hh:mm" returns "03:16" where "HH:MM" returns " 3:16", your choice.

See also

CurrentDate, CurrentDateTime, CurrentTime, Number

Example

```
Rem Different demos of the Format feature

Dim tDate As Date, tTime As Time, tDateTime As DateTime
Dim tStr As String
CurrentTime(tTime)
CurrentDate(tDate)
CurrentDateTime(tDateTime)
Print FormatTime(tTime, "hh:mm:sst"),
Print FormatDate(tDate, "dd/mm/yyyy")
Print FormatTime(tDateTime.T, "hh:mmte"),
Print FormatDate(tDateTime.D, "dd/mm/yy")
tStr = "The time is now : " + \
    FormatTime(tDateTime.T, "hh:MM:SS")
Print tStr
```

FREEFILE Function

Return the next available file number.

Syntax

FreeFile {As Integer}

Remarks

Rather than trying to keep track of file numbers or **FreeFile** will return the next available number for you to use. **FreeFile** should not be used alone but its value should be assigned to a variable. While it is possible to open a file directly with the **FreeFile** function only the **Close** statement without a file number can be used to close it.

If there are no available file numbers then **FreeFile** returns a -1 value so a check should be made to ensure that this function has supplied a valid file number for use.

See also

Open, Close

Example

```
Rem Make a copy of a text file
Sub ExitNoFiles
   Print "No more files available. Halting!"
   Delay(3)
   Reset
   End : Rem Make a swift and abrupt halt here
End Sub
Dim InputFile As Integer, OutputFile As Integer, tStr As String
InputFile = FreeFile
If InputFile > 0 Then : Rem Assume file exists
   Open "Message.TXT" For Input As #InputFile
Else ExitNoFiles
```

```
End If
OutputFile = FreeFile
If OutputFile > 0 Then
    Open "CopyMsg.TXT" For Output As #OutputFile
Else ExitNoFiles
End If
While Not(EOF(InputFile))
    Input #InputFile, tStr
    Print #OutputFile, tStr
Wend
Close #InputFile
Close #OutputFile
```

FUNCTION Statement

Define a subprogram that will return a value.

Syntax

Function *Name*[(*Arg* As Type[, *Arg* As Type]...)] As Type

Statements...

Name = *ExpressionOfDeclared* Type

End Function

Remarks

A **Function** is a subroutine or sub-program that performs a specific task which returns a value. While it is similar in structure to a **Sub** the main difference is that it is declared as a data type and can therefore be used to return a value of this type.

You would most often use a **Function** if you found that you were typing the same or very similar code, two or more times in your source code. The preferred approach in such a case is to write a **Function** that will perform the task and return a value simply by calling it rather than duplicating code.

The *Name* is the name you assign to the function. The argument(s) *Arg* is the name of a formal argument to the function usually referred to as parameters. Each argument in the parameter list must have an associated type declaration using the **As** keyword and each must be separated by a comma. If parameters are defined in a function header they must be enclosed in parentheses.

You may define local variables within your user-defined **Function** and you can also use **Static** declarations to preserve the value of individual variables across **Function** calls. Normally any variables defined in a subroutine or function are discarded on exiting but the use of the **Static** declaration will maintain the value of these specific variables while the program is running.

A **Function** can optionally have a value of its declared type assigned to it for the caller to use. See the following example as it is easier to depict the process rather than describe it. Assigning a value to a **Function** can be done at any point as long as it is set prior to exiting the **Function**. If no value is assigned to the **Function** it will return zero in the case of a numeric **Type** definition and a string of zero length for a string **Type** declaration.

It is an accepted practice to define a default value to a **Function** result as the first line in its source code, then if exiting early, the value is already set. An example of this is shown in the QuickExit **Function**.

Note!

Functions *cannot* be defined within another function definition.

For a discussion on the use of a varying number of parameters, refer to the preceding chapter under the sub-heading of Subs and Functions.

See also

Declare, Exit, Static, Sub

Example

```
Rem Define functions and demo the Static declaration

Function QuickExit(IntA As Integer) As Integer
   QuickExit = False : Rem Set the most likely result
   If InputYesNo("Counter = " + Str(IntA) + " exit anyway? ") Then
      QuickExit = True
   End If
End Function
Function CapName(tStr As String, TimesUsed As Integer) As String
   Dim CapStr As String
   Static NamesCapped As Integer
   CapStr = LCase(tStr) : Rem Copy to not change the original
```

```
    CapName = UCase(Left(CapStr,1)) + Mid(CapStr,2,Len(CapStr)-1)
    NamesCapped = NamesCapped + 1
    TimesUsed = NamesCapped
End Function

Dim NameStr As String, Ctr As Integer
Ctr = 0 : Rem Not needed, but on the subject of good practice!
While Ctr < 3
    Print "Enter your name "; : Input NameStr
    Print "Hello "; CapName(NameStr,Ctr)
    Print "The Function CapName has now been used "; Ctr; " times"
    If QuickExit(Ctr) Then End
Wend
```

GET Statement

Read information from a file.

Syntax

Get [#]*FileNum*, [*Position*], *Name*

Remarks

The *FileNumber* is the number assigned to the open file.

The *Position* argument is the number of the record in a **Random** access file, or the number of the byte in a **Binary** file. Note that unlike some other programming languages the first record or byte in the file is number 1, not number 0.

If the *Position* is not specified, then the record is read from the current file position but note that a comma must be used as a place-holder for the absent *Position* value.

The *Name* is the name of the variable that receives the returned data.

Note!

If you are using **Get** to read a record from a **Random** access file you must ensure that the variable is of the correctly declared type that was used to **Put** the data in the file in the first place. Failing to do so could lead to some very interesting-looking data.

See also

FreeFile, Input, Open, Put, Type...End Type

Example

```
Rem Assume 'Records.DAT' exists with 'UserRec' structure and data

Type UserRec
   Name As String*30
   Alias As String*30
   LastDate As String*8
   TimesCalled As Integer
End Type
Dim DataF As Integer, URec As UserRec
DataF = FreeFile          : Rem Assume file exists
Open "Records.DAT" For Random As #DataF Len = Len(UserRec)
While Not(EOF(DataF))
   Get #DataF,, Urec     : Rem Step sequentially from
   Print LeftPad(URec.Name,32); LeftPad(URec.Name,32);
   Print LeftPad(URec.LastDate,10); URec.TimesCalled
Wend
Close #DataF
```

GETCOLOR Function

Return the current screen color.

Syntax

GetColor {As Integer}

Remarks

When using the *Wildcat!* @Codes (pronounced At Codes) for colors the screen will continue to use the last color defined by the @Codes until changed to another color either specifically or by using **DefColor** in a **Print** statement.

Using **GetColor** will return the decimal value of the current screen color but to convert it to the @Code counterpart, you need to get the **Hex** value of it and add the @-signs either side.

See the Example-two program in the concatenation command for more information on how the use of @Codes can be affected.

See also

DefColor

Example

```
Rem Demonstrate a range of colors using @Codes

Const RedYellow = "@E4@"
Const BlueWhite = "@F1@"
Dim HexVal As String
Print RedYellow + "Fried eggs and Ketchup"
Print BlueWhite; : Print "Sailors ahoy!"
HexVal = Right(Hex(GetColor),2)
Print "We are printing in @"; HexVal; "@ color at the moment."
Print DefColor; "Finally black to sanity."
```

GETCONFINFO Function

Get the details of a conference.

Syntax

GetConfInfo(*ConfRec*, *ConfNum*) {As True/False}

Remarks

This command will transfer the *ConfNum* conference details into the *ConfRec*. The *ConfNum* is an integer within the range of your conferences and *ConfRec* is a record of the internal data type **ConfRecord**.

See also

GetFileArea

Example

```
Rem Attempt to retrieve a conference record

Dim tConfRec As ConfRecord, tConfNum As Integer
tConfNum = InputNumber("Conference number:", 0, \
                MakeWild.MaxConfAreas -1)
If GetConfInfo(tConfRec, tConfNum) Then
   Print "This conference is "; tConfRec.Name
Else
   Print "Conference request error!"
End If
```

GETFILEAREA Function

Get the details of a file area.

Syntax

GetFileArea(*FileRec*, *FileAreaNum*) {As True/False}

Remarks

This function will return the *FileRec* record for the *FileAreaNum*. It returns **True** or **False** on the success of the operation. The *FileRec* is a record of the internal data type **FileAreaRecord**.

See also

GetConfInfo, GetFileInfo

Example

```
Rem Display information on all file areas

Dim FRec As FileAreaRecord, AreaNum As Long
AreaNum = 0
While GetFileArea(FRec, AreaNum)
    Print FRec.Name
    Print FRec.Path
    AreaNum = AreaNum + 1
Wend
```

GETFILEINFO Function

Get file record information.

Syntax

GetFileInfo(*FileRec*, *FileName*) {As True/False}

Remarks

This command will retrieve the *FileRec* file information for the file *FileName* if it exists. The *FileName* can be a string variable or a literal enclosed in quotes. This function will return either **True** or **False** on the success of the mission. The *FileRec* is a record of the internal data type **FileRecord**.

If you wanted to step through the entire database then set *FileName* to a null string ("") in order to retrieve the first file.

See also

AddFile, GetFileArea, GetNextFile, UpdateFile

Example

```
Rem List all files from a start point in the database

Dim fRec As FileRecord, tStr As String
Print "Starting file to retrieve. [Quit to exit] : ";
Input tStr
If UCase(tStr) <> "QUIT" Then
   If GetFileInfo(fRec, tStr) Then
      Do
         Print "        File: "; fRec.Name
         Print "Description: "; fRec.Description
      Loop Until Not(GetNextFile(fRec))
   End If
End If
```

GETMESSAGE Function

Get the message header details from the message database.

Syntax

GetMessage(*MsgHdr*, *MsgNum*) {As True/False}

Remarks

This command will retrieve the message header details for *MsgNum* and place them in the *MsgHdr*. The *MsgNum* is an integer within the range of the available messages in the currently selected conference and the *MsgHdr* is a record of the internal data type **MessageHeader**.

This function will return either **True** or **False** on the success of the attempt.

See also

GetMessageText, GetNextMessage, Join, UpdateMessage

Example

```
Rem Retrieve a known message number

Dim Msg As MessageHeader, tInt As Integer
tInt = InputNumber("Message number: (9999 to Quit) ",0,9999)
If tInt <> 9999 Then
   If GetMessage(Msg, tInt) Then
      Print "Subject: "; Msg.Subject
   End If
End If
```

GETMESSAGETEXT Function

Copy the message text for a particular message to a text file.

Syntax

GetMessageText(*MsgNum*, *FileName*) {As True/False}

Remarks

This function will retrieve the message text for *MsgNum* in the current conference and copy it to the text file *FileName*. If *Filename* exists, it will be overwritten without warning.

This function returns **True** or **False** on the success of the operation.

See also

GetMessage, GetNextMessage

Example

```
Rem Read and display the message text for a given message number

Dim MsgNum As Integer, tStr As String
Const FName = "TempFile.TXT"
Print "Message to view: ";
Input MsgNum
If GetMessageText(MsgNum, TempPath+ FName) Then
   Open TempPath+ FName For Input As #1
   While Not EOF(1)
      Input #1, tStr
      Print tStr
   Wend
   Close #1
End If
```

GETNEXTFILE Function

Get the next file in sequence from the files database.

Syntax

GetNextFile(*FileRec*) {As True/False}

Remarks

This command works in conjunction with **GetFileInfo**. After **GetFileInfo** has retrieved the first requested file then **GetNextFile** will continue through the database until no more files remain. The *FileRec* is a record of the internal data type **FileRecord**.

This function will return either **True** or **False** on the success of the attempt.

See also

AddFile, UpdateFile

Example

```
Rem List all files from a start point in the database

Dim fRec As FileRecord, tStr As String
Print "File header to retrieve. [Quit to exit] : ";
Input tStr
If UCase(tStr) <> "QUIT" Then
   If GetFileInfo(fRec, tStr) Then
      Do
         Print "        File: "; fRec.Name
         Print "Description: "; fRec.Description
      Loop Until Not(GetNextFile(fRec))
   End If
End If
```

GETNEXTMESSAGE Function

Get next message in sequence from the message database.

Syntax

GetNextMessage(*MsgRec*) {As True/False}

Remarks

This command works in conjunction with GetMessage. After GetMessage has retrieved the first requested message then GetNextMessage will continue through the database until no more messages remain. The *MsgRec* is a record of the internal data type MessageHeader.

This function will return either True or False on the success of the attempt.

See also

GetMessage, Join, UpdateMessage

Example

```
Rem Display the subject heading for this conference

Dim Msg As MessageHeader, tInt As Integer
tInt = InputNumber("Message number: (9999 to Quit) ",0,9999)
If tInt <> 9999 Then
   If GetMessage(Msg, tInt) Then
      Do
         Print "Subject: "; Msg.Subject
      Loop Until Not GetNextMessage(Msg)
   End If
End If
```

GETNEXTUSER Function

Get the next user in sequence from the user database.

Syntax

GetNextUser(*UserRec*) {As True/False}

Remarks

This command works in conjunction with **GetUser**. After **GetUser** has retrieved the first requested user then **GetNextUser** will continue through the list until no more users remain. The *UserRec* is a record of the internal data type **UserRecord**.

This function will return either **True** or **False** on the success of the attempt.

See also

GetUser, AddUser, UpdateUser

Example

```
Rem Starting from a known position display all users

Dim UserRec As UserRecord, tStr As String
Rem A tStr = "" will start at the beginning of database
Print "Starting user name [Quit to exit] : ";
Input tStr
If UCase(tStr) <> "QUIT" Then
   If GetUser(UserRec, tStr) Then
      Do
         Print "Name: "; UserRec.Name; Tab(30); UserRec.From
      Loop Until Not GetNextUser(UserRec)
   End If
End If
```

GETNODEINFO Statement

Get a specific node information record.

Syntax

GetNodeInfo *NodeRec, NodeNum*

Remarks

This command will retrieve the node information for the *NodeNum* node number and store it in *NodeRec*. The *NodeRec* is a record of the internal data type **NodeInfoRecord** and NodeNum is an integer in the range from zero to **MaxNode**.

Example

```
Rem Step through the nodes, display security profiles

Dim nRec As NodeInfoRecord, NodeNum As Integer
Do
    NodeNum = InputNumber("Node: (999 = Quit) ", 0, 999)
    If NodeNum <> 999 Then
        GetNodeInfo nRec, NodeNum
        Print "Node "; NodeNum; " Security "; nRec.Security
    End If
Loop Until NodeNum = 999
```

GETSECPROFILE Function

Get a specific security profile.

Syntax

GetSecProfile(*SecRec*, *ProfileName*) {As True/False}

Remarks

This command will retrieve the security profile for the *ProfileName* profile and store it in *SecRec*. The *SecRec* is a record of the internal data type **SecurityProfile** and *ProfileName* is a valid security profile name in string format. The *ProfileName* may be a string variable or a literal enclosed in quotes.

This function will return either **True** or **False** on the success of the attempt.

Example

```
Rem Retrieve a security profile and display the results

Dim sRec As SecurityProfile
If GetSecProfile(sRec, "SYSOP") Then
   Print "Name: "; SRec.Name
   Print "Expired name: "; sRec.ExpiredName
End If
```

GETUSER Function

Get a specified user's details.

Syntax

GetUser(*UserRec*, *UserName*) {As True/False}

Remarks

This command will retrieve a node user's details and store them in *UserRec*. The *UserRec* is a record of the internal data type **UserRecord** and the *UserName* is a string. If you wanted to step through the entire database then set *UserName* to a null string ("") in order to retrieve the first user name.

This function will return either **True** or **False** on the success of the attempt.

See also

GetNextUser, UpdateUser

Example

```
Rem Display information on a single user

Dim UserRec As UserRecord, tStr As String
Print "User name [Quit to exit] : "; : Input tStr
If UCase(tStr) <> "QUIT" Then
   If GetUser(UserRec, tStr) Then
      Print "Name: "; UserRec.Name; Tab(30); UserRec.From
   Else
      Print "User name ["; UCase(tStr); "] was not found"
      Beep
   End If
End If
```

GOODBYE Statement

Gently log off a caller.

Syntax

Goodbye

Remarks

This command is exactly the same as selecting the [G]..Goodbye menu option. If the caller has novice menus turned on, they will have the option of aborting the logoff process, in which case the program will simply continue.

See also

HangUp

Example

```
Rem Gently logoff the caller

GoodBye
```

GOSUB Statement

Branch unconditionally to the specified line number or label.

Syntax

GoSub {*Line* | *Label*}

Statements...

Return

Remarks

The *Line* and *Label* indicators refer to the line number or label to which the program flow is to be directed. Program execution will continue from this point until a **Return** statement is encountered at which point program flow will resume at the statement immediately following the originating **GoSub** instruction.

You assign a name for the *Label* which must end with a colon to indicate it is a label. Do not confuse this trailing colon with the use of the colon to separate several program statements on the same line of code.

Labels are ignored by the program flow unless they are specifically called on to branch to by a **GoSub** or **GoTo** statement

If contained within a subroutine or function the **GoSub** *Label* instructions are local to that subroutine or function which means it is not possible to branch out of that construct.

The *Label* can be positioned anywhere in the source code either before or after the calling **GoSub** instruction providing it is within the local structure of a subroutine or function, or in the main body of code. Unlike **Sub** and **Function** it does not require a **Declare** statement if called before its description.

The **Sub** and **Function** constructs provide a much better method of supporting subprograms than **GoSub**...**Return**. The debugging process can be very confusing with multiple **GoSub**...**Return** statements.

Note!

The **End** statement in the example code is necessary as program flow would continue past the Label if it were not present.

See also

Function, GoTo, Sub

Example

```
Rem Branching forward/backwards with the GoSub/GoTo

Print "I should always use Sub and Function for branching"
GoSub BadMove
Print "Progam now terminating with the End statement"
End
Rem End of the main program body
BadMove:
   Dim YesSir As String*1
   Print "Do you want to see more of this ";
   Input YesSir
   If UCase(YesSir) = "N" Then
      Return
   Else
      GoTo BadMove
   End If
Rem No Return needed here as the If statement takes care of it
```

GOTO Statement

Branch unconditionally to a specified line or label.

Syntax

GoTo {*Line* | *Label*}

Remarks

The *Line* and *Label* indicators refer to the line number or label to which the program flow is to be directed. Program execution will continue from this point. Unlike **GoSub** there are no associated **Return** statements to use with **GoTo**.

You assign a name for the *Label* which must end with a colon to indicate it is a label. Do not confuse this trailing colon with the use of the colon to terminate several program statements on the same line of code. Labels are ignored by the program flow unless they are specifically called to branch to by a **GoSub** or **GoTo** statement

If contained within a subroutine or function the **GoTo** *Label* instructions are local to that subroutine or function which means it is not possible to branch out of that construct.

The *Label* can positioned anywhere in the source code either before or after the calling **GoTo** instruction providing it is within the local structure of a subroutine or function, or in the main body of code. Unlike **Sub** and **Function** it does not require a **Declare** statement if called before its description.

The **Sub** and **Function** constructs provide a much better method of supporting subprograms than **GoTo** and these should be used unless absolutely necessary. The debugging process can be very confusing with multiple **GoTo** statements.

Unlike **GoSub** there are rare occasions when the **GoTo** *Label* can simplify a branching decision for program flow but it is highly recommended that you try all other approaches before resorting to the **GoTo** statement.

Note!

The **End** statement in the example code is necessary as program flow would continue past the Label if it were not present.

See also

Function, GoSub, Sub

Example

```
Rem Branching forward/backwards with the GoSub/GoTo

Print "I should always use Sub and Function for branching"

GoSub BadMove

Print "Progam now terminating with the End statement"

End
Rem End of the main program body


BadMove:
   Dim YesSir As String*1

   Print "Do you want to see more of this ";

   Input YesSir

   If UCase(YesSir) = "N" Then
      Return Else GoTo BadMove

   End If
Rem No Return needed here as the If statement takes care of it
```

HANGUP Statement

Hang up the modem without warning.

Syntax

HangUp

Remarks

The HangUp command will attempt to immediately hang up the phone connection without notification to the caller.

See also

Break, Carrier, Goodbye

Example

```
Rem Hang up if incorrect password tried

Dim Ctr As Integer, PWordOk As Integer, PWordStr As String
Do
   Ctr = Ctr + 1
   Print "Enter your password ";
   Input PWordStr
   PWordOk = (UCase(PWordStr) = "SYSOP")
Loop Until (Ctr > 3) Or (PWordOK)
If Ctr > 3 Then
   Print "Illegal access attempted."
   If Not(Local) Then HangUp
End If
```

HELLOFILES Statement

Display the Hello files to a caller.

Syntax

HelloFiles

Remarks

The **HelloFiles** statement is intended for use in a logon program or similar for a new user to the system. It presents the exact same response as pressing the [I]..Initial welcome selection on the menu.

For **HelloFiles** to work there must be at least one file in the DISP directory with a name in the range of HELLO1.BBS through HELLO9.BBS with the hello information in them. **HelloFiles** will scan and display as many of the 9 files that exist.

Example

```
Rem Greet caller with Hello files, then questionnaire #1

HelloFiles
Questionnaire 1
```

HELPLEVEL Statement

Set a caller's help level.

Syntax

HelpLevel

Remarks

The **HelpLevel** statement presents the user with the same response as if [H]..Help level had been selected from the menu. Once selected the user can change their help level to display full menus or just the cryptic one-liners.

Example

```
Rem Allow the user to reset the Help level

If InputYesNo("Tired of cryptic menus? ") Then
   HelpLevel
End If
```

HEX Function

Convert a decimal value to its Hexadecimal equivalent.

Syntax

Hex(*Val*) {As String}

Remarks

The *Val* is the numeric variable or expression the **Hex** function will convert into a **String** type. This value may be any numeric type which will be converted to either integer or a long integer representation as appropriate with any fractional values ignored.

Hex returns a clean string of Hexadecimal characters which means it does not have an identifier to say that it is a Hexadecimal value. When using the **Val** function Hexadecimal value strings have a dollar sign as the leading character in the string to identify them as such when being used in a conversion or calculation process.

The **Val** function can be used to convert Hexadecimal **String** values back into decimal by adding a dollar sign in front of the Hexadecimal string. There must be no spaces between the dollar sign and the Hexadecimal string value otherwise **Val** will return 0.

The **Val** function is currently the only instance where the dollar sign can be used in *wcCODE* for converting Hexadecimal values to their decimal equivalents.

See also

Oct, Val

Example

```
Rem Demo the use of Hex
Dim HexStr As String : HexStr = Hex(90210)
Print "Hollywood hits hex = "; HexStr
Print "Doing decimal deal = "; Val("$"+HexStr)
Print "The Hex value of 123 is "; Right(Hex(123),2)
```

IF Statement

Make a decision on program flow.

Syntax

If *Condition* Then

Statements...

[ElseIf *PartCondition* Then

Statements...]

[Else

Statements...]

End If

Remarks

Exp is an expression that yields a result of **True** or **False** which can include any combination of valid expressions grouped with the Boolean operators **And**, **Eqv**, **Imp**, **Not**, **Or**, **Xor**. Also any expression which can resolve to a yes or no answer using any of the equality/inequality operators <, >, =, <>, <=, >= can also be used in an **If...Then...** statement.

The **Then** portion is a set of statements to be executed if *Exp* meets the 'true' part of the decision. Similarly the **Else** portion is a set of statements to be executed if *Exp* equates to false.

If the first condition resolves to false then program flow will branch to the statements following **Else.**

The **ElseIf** is a second chance option for a evaluation of the **If...Then** portion of the argument. If the full evaluation resolves to **False** then **ElseIf** can verify that some part of the condition may be true and would satisfy an alternate to both 'true' or 'false' by providing intermediate tiers of code.

The **ElseIf** option does *not* have to act on the original *Condition* but it makes sense that if it is included in the current **If...Then** structure that a

Condition based on the original would be logical. What this means is that it is quite okay to have **If** A > B **Then**...**ElseIf** C > D **Then**... but the logic of the original statement is lost and the results could be confusing.

Else and **ElseIf** are optional as a single conditional branch may be all that is required for program flow. The use of **If...Then** can be a valid statement where no **End If** is required only if the entire **If** *Condition* **Then** *Statement* is contained on a single line of source code. Once you break the line even for a single statement after the keyword **Then**, an **End If** statement becomes mandatory.

See also

Do...Loop, Else, Select Case

Example

```
Rem Demonstrate If, Then, Else and ElseIf

Dim NameStr As String, DOB As String
Dim TodayStr As String, Today As Date
Print "Enter your name "; : Input NameStr
Do
   Print "Enter your DOB (mm/dd) "; : Input DOB
Loop Until Len(DOB) = 5
CurrentDate Today
TodayStr = FormatDate(Today, "mm/dd")
If Instr(DOB, TodayStr) = 1 Then
  Print "Happy birthday "; NameStr
ElseIf Instr(DOB,Left(TodayStr,2)) = 1 Then
  Print "Well "; NameStr; " at least it's this month."
Else
  Print "Patience is a game and a virtue "; NameStr
End If
```

IMP Operator

Perform a logical Implication operation between operands.

Syntax

Op1 Imp *Op2*

Remarks

The **Imp** operator performs an implication operation between the two operands, *Op1* and *Op2*. This Boolean function has fairly obscure uses or applications within *wcCODE* and it is only included as it is a valid Boolean operator.

The *Op1* and *Op2* operands are logical expressions or numeric values of any valid data type of **Byte**, **Integer**, or **Long**.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

And, Eqv, Not, Or, XOr

Example

```
Rem Bitwise implication example

Dim IntA As Integer, IntB As Integer
IntA = 25
IntB = 31
Print "The implication here is "; IntA Imp IntB
```

INCOMM Function

Read a character from the communications port

Syntax

InComm {As String}

Remarks

This function will read a character from the communications port if any are pending. It is similar to the **Inkey** function but unlike **Inkey**, it only reads input from the communications port.

See also

Inkey

Example

```
Rem Read data from the COM port
Dim tStr As String, KeyIn As String*1
Dim TimeOut As Integer, TimeHolder As Real
tStr = ""
Do
    TimeHolder = Timer + 3
    Do
        KeyIn = InComm
        TimeOut = (Timer = TimeHolder)
    Loop Until (KeyIn > "") Or (TimeOut)
    If TimeOut Then Exit Do
    If KeyIn <> Chr(13) Then
        tStr = tStr + KeyIn
    End If
Loop Until KeyIn = Chr(13)
If TimeOut Then
    Print "Timed out waiting on COM port"
Else
    Print "Received: "; tStr
End If
```

$INCLUDE Directive

Include another source code file.

Syntax

'$Include *FileName*

Remarks

The **'$Include** directive is an instruction to the compiler to include the source code contents of the *FileName* of *wcCODE* into the current file. This command differs from the **Chain** and **Run** commands in that it physically includes the contents of *FileName* during the compilation process.

The ability to include files is especially useful if you have a standard set of common subroutines and functions that you use in other *wcCODE* programs and want to include them in the current one too.

The *FileName* file must exist in the current directory or a full drive and path must be provided. The file extension is mandatory and must be included.

It is possible for the *FileName* file to also use the **'$Include** directive to nest other source code files within itself. You are limited only by the available memory in your system as to how deep this nesting can be taken.

You will notice the leading apostrophe that is part of the syntax of this command is also the comment marker for *wcCODE*. To comment out a line of code with the **'$Include** directive in it, you will need to add another apostrophe, or the keyword **Rem** in front of the statement.

Note!

A word of caution. Nesting **'$Include** files deeply can make the debugging process a little more involved as you chase from one file to another trying to discover the errant bug in your code.

See also

Chain, Run

Example

```
Rem MyProg.WCC - This is the main program file

Rem Include standard utility files
'$Include "MyUtils.WCC"
Dim NameStr As String
Input "Enter your first name "; NameStr
Print "Hi "; CapName(NameStr)
Rem End of the main program body

Rem Save as MyUtils.WCC - This file will be included

Function CapName(tNameStr As String) As String
   Dim TempStr As String
   TempStr = LCase(tNameStr) : Rem first convert to all lower case
   tNameStr = UCase(Mid(TempStr, 1, 1))
   tNameStr = tNameStr + Mid(TempStr, 2, Len(TempStr) - 1)
   CapName = tNameStr
End Function
```

INDEXADD Statement

Add a new index key to the indexing system.

Syntax

IndexAdd *String*, *RecordPos*

Remarks

This command will add a new key *String* to the index system and store it with the *RecordPos* positioning information for its later retrieval. Generally the RecordPos will be a file record number in a Random file. The length of the String must be greater than zero as the retrieval functions cannot work with zero-length keys.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexClose, IndexLookup, IndexNextKey, IndexNextRef, IndexOpen, IndexQuery, IndexQueryNext

Example

```
Rem Assume the datafile exists

Dim tStr as String, Ctr As Long, tRec As UserRecord
Open "TestData.DAT" For Random As #1 Len = Len(UserRecord)
IndexOpen("TestData.IDX")
While Not(EOF(1))
    Ctr = Ctr + 1
    Get #1, Ctr, tRec
    tStr = Trim(tRec.Zip)
    IndexAdd(tStr, Ctr)
Wend
```

INDEXCLOSE Statement

Close the currently open index file.

Syntax

IndexClose

Remarks

This statement when called, will close the currently open index system. If no index has been opened then no error is returned.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, IndexLookup, IndexNextKey, IndexNextRef, IndexOpen, IndexQuery, IndexQueryNext

Example

```
Rem Integral part of the Indexing system

IndexClose : Rem Declares no error if no index open
```

INDEXLOOKUP Function

Initialize the index-retrieval system with a starting string.

Syntax

IndexLookUp(*String*) {As String}

Remarks

This command will use the current open index to perform a lookup function for the key *String*. The string value returned by **IndexLookUp** will be the key that was located. The *String* must be at least one character in length for **IndexlookUp** to function correctly.

If an exact match is found, then **IndexLookUp** will also equal *String*. If no exact match is found, then the next highest value part-matching *String* will be returned. For instance if an index contained the keys "QUACK", "QUICK", "QUIET" and "QUIT" and *String* contained "QUID" it would return a zero length string. However if *String* contained "QUI" then it would return "QUICK" as the next highest partial match.

Use **IndexNextRef** and **IndexNextKey** to process the index in sequence after the call to **IndexLookUp**.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, IndexClose, IndexNextKey, IndexNextRef, IndexOpen, IndexQuery, IndexQueryNext

Example

```
Rem Assume the data and index files already exist

Dim tStr1 As String, RecPos As Long
Dim tStr2 As Sting, tRec As UserRecord
tStr1 = "WIL"
Open "Testdata.DAT" For Random as #1 Len = Len(UserRecord)
IndexOpen("TestData.IDX")
tStr2 = IndexLookUp(tStr1)
If tStr1 = tStr2 Then : Rem Retrieve only exact matches
   Do
      RecPos = IndexNextRef
      If RecPos <> -1 Then
         Get #1, RecPos, tRec
         Print tRec.Name, tRec.Zip
      End If
   Loop Until RecPos = -1
End If
Close #1
IndexClose
```

INDEXNEXTKEY Function

Return the next key.

Syntax

IndexNextKey {As String}

Remarks

The **IndexNextKey** function will return the next key in the sequence after **IndexNextRef** has found all of the exact matches to the current key. If no further keys are available then the function will return a zero length string. As with **IndexNextRef**, this function must be initialized by a call to **IndexLookUp**.

This function does not return an index position to the data, it merely locates a starting point for the indexing system to begin from. The positioning data is retrieved by calling **IndexNextRef**.

The **IndexNextKey** function will continue to return keys that contain the original data passed to **IndexLookUp**. This means that if a key of "C" was passed to **IndexLookUp**, then **IndexNextKey** will continue to return all of the keys that start with the letter "C". It will return a null string when it encounters the next higher letter in the index.

To traverse an index alphabetically you would need to call **IndexLookUp** with each letter of the alphabet before calling the **IndexNextRef** and **IndexNextKey** sequences. See **IndexNextRef** for an example of this process.

For an another example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, **IndexClose**, **IndexLookup**, **IndexNextRef**, **IndexOpen**, **IndexQuery**, **IndexQueryNext**

Example

```
Rem Assume the index and data already exist

Dim tStr1 As String, rPos As Long, uRec As UserRecord
tStr1 = "WILD"
Open "TestData.DAT" For Random As #1 len = Len(UserRecord)
IndexOpen("TestData.IDX")
If Instr(IndexLookUp(tStr1),tStr1) <> 0 Then
  While tStr1 <> ""
     Do
        rPos = IndexNextRef
        If rPos <> -1 Then
           Get #1, rPos, uRec
           Print "Found "; uRec.Name
        End If
     Loop Until rPos = -1
     tStr1 = IndexNextKey
  Wend
End If
Close #1 : IndexClose
```

INDEXNEXTREF Function

Return the positioning information for the next exact-match key.

Syntax

IndexNextRef {As Long}

Remarks

This command is used in conjunction with **IndexLookup** and **IndexNextKey**. In fact, **IndexLookUp** must be called before **IndexNextRef** can work. The value returned will be the positioning data originally passed to the indexing system with the call to **IndexAdd**.

Normally the data would be stored in a data file and the positioning data would be the record number. The data could just as easily be stored in an array. When no further matches to the data can be found then **IndexNextRef** returns a value of negative-one.

The example program presented here, will demonstrate the use of this function better than a lengthy description. You will need to assume that the data file assigned as #1, the record tRec and the index file already exist.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, **IndexClose**, **IndexLookup**, **IndexNextKey**, **IndexOpen**, **IndexQuery**, **IndexQueryNext**

Example

```
Rem Assume Index and data file exist

Dim tStr As String, Ctr As Integer
Dim RecPos As Long, tRec As UserRecord
Open "Testdata.DAT" For Random As #1 Len = Len(UserRecord)
IndexOpen("TestData.IDX") : Ctr = 1
Do
   tStr = Chr(Ctr + 65) : Rem a full alphabetical search
   tStr = IndexLookUp(tStr)
   While tStr > ""
      Do
         RecPos = IndexNextRef
         If RecPos > -1 Then
            Get #1, RecPos, tRec
         End If
      Loop Until RecPos = -1
      tStr = IndexNextKey : Rem More keys left?
   Wend
   Ctr = Ctr + 1
Loop Until Ctr = 26
Close #1 : IndexClose
```

INDEXOPEN Function

Open an index file and initialize the index system.

Syntax

IndexOpen(*FileName*) {As True/False}

Remarks

This command will open the index file, *FileName* and return either **True** or **False** on the success of the operation. If the file does not exist, it will be created ready for use with **IndexAdd**. There is no default extension for the index file, so you should provide one at this stage.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, IndexClose, IndexLookup, IndexNextKey, IndexNextRef, IndexQuery, IndexQueryNext

Example

```
Rem Function to open an index with a correct file extension

Function OpenIndex(FName As String) As Integer
   Dim tFName As String
   Const IdxExt = ".IDX"
   tFName = UCase(FName)
   If (Instr(tFName, IdxExt) = 0) And \
       (Instr(tFName, ".") = 0) Then
         tFName = tFName + IdxExt
         OpenIndex = IndexOpen(tFName)
   End If
End Function
```

```
If OpenIndex("TestFile") Then
   Rem Plug in your indexed processing code here
Else
   Print "Could not open the index file"
   Beep
End If
```

INDEXQUERY Function

Apply Boolean logic to a multiple key search.

Syntax

IndexQuery(String) {As True/False}

Remarks

This function accepts Boolean operators **And**, **Or**, and **Not** within *String* to look for items in an index. It will return **True** if there are one or more items to be retrieved that match the search criteria. On the function returning **True**, the **IndexQueryNext** function is called to retrieve the positioning data.

The data to be searched for can *not* contain any of the three operators (**And**, **Or**, **Not**) as part of the key-text otherwise they will be interpreted as operators. For example if you had a key of "Jim and me", then **IndexQuery** would treat the "and" as an operator and return a result based on both "Jim" and "me" occurring in the index file. Since the index system looks only at the left-most characters, the "me" in "Jim and me" would never be found.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, IndexClose, IndexLookup, IndexNextKey, IndexNextRef, IndexOpen, IndexQueryNext

Example

```
Rem Retrieve keys with BBS and GOOD, but not MAJOR in them

Dim RecPos as Long, tRec As UserRecord
Open "Testdata.DAT" For Random As #1 Len = Len(UserRecord)
IndexOpen("TestData.IDX")
```

```
If IndexQuery("BBS And GOOD And Not MAJOR") Then
   Do
      RecPos = IndexQueryNext
      If RecPos <> -1 Then
         Get #1, RecPos, tRec
         Print "User "; tRec.name
      End If
   Loop Until RecPos = -1
End If
Close
```

INDEXQUERYNEXT Function

Return the positioning information for the keys.

Syntax

IndexQueryNext {As Long}

Remarks

This function works in conjunction with **IndexQuery**. Once **IndexQuery** indicates there are matches to be retrieved, then **IndexQueryNext** will continue to return positioning information for the data until the search request for **IndexQuery** is exhausted. To indicate the end of the search **IndexQueryNext** will return a value of negative one.

For an example of the index system at work, see the sample program WCLIST.WCC and its associated files. The sample programs are briefly described in the Appendix at the back of this manual.

See also

IndexAdd, IndexClose, IndexLookup, IndexNextKey, IndexNextRef, IndexOpen, IndexQuery

Example

```
Rem Retrieve keys with BBS and GOOD, but not MAJOR in them

Dim RecPos as Long, tRec As UserRecord
Open "Testdata.DAT" For Random As #1 Len = Len(UserRecord)
IndexOpen("TestData.IDX")
If IndexQuery("BBS And GOOD And Not MAJOR") Then
   Do
      RecPos = IndexQueryNext
      If RecPos <> -1 Then
         Get #1, RecPos, tRec
         Print "User "; tRec.name
      End If
   Loop Until RecPos = -1
End If
Close
```

INKEY Function

Read a character from the keyboard, communications port, or file.

Syntax

Inkey [(*FileNum*)] {As String}

Remarks

The **Inkey** function scans the keyboard buffer the communication port or optionally, a text file for any characters waiting to be read. In all cases, if a character is pending, **Inkey** will read it in as a **String** value. It is then up to your program to accurately identify value of the returned string as some keycodes are handled differently to straight alpha-numeric characters.

The Inkey function when scanning the keyboard will return the character zero (ASCII 0) if an extended key has been pressed. An extended key is defined as any of the function keys F1 through F10 and F11 and F12 if the last two exist on your keyboard. Also the cursor keys and six other keys, 'Ins', 'Del', 'Home', 'End', 'PgUp', 'PgDn' are all classified as extended keys.

In the case of an extended code, **Inkey** will still contain another keystroke to be read. This next keystroke while it will have an ASCII value will actually be the ASCII equivalent of the scan code for the key pressed.

Each key on the PC keyboard has a unique scan code which is normally translated into the character value for that key. However the handling of the extended keys presents a slightly different approach, hence the need to read **Inkey** the second time to return the appropriate key value.

By providing **Inkey** with the optional *FileNum*, **Inkey** can read characters from a file that has been opened with the same *FileNum*.

Note!

The extended keys F1 through F10 are reserved for *Wildcat!*'s use and cannot be trapped with the **Inkey** function. These keys can however be used in conjunction with the ALT, CTRL and SHIFT key combinations.

Another extended key that is not available to **Inkey** is the ↑ as it is used by *Wildcat!* as the Scroll-back feature for the Sysop to review activities scrolled off the screen.

For another example of the use of the **Inkey** function and the requirements of your program to translate the extended keys, see the **MoveRight** command.

See also

InComm, Input, InputYesNo, WaitEnter

Example

```
Rem Make a response file and then use it with Inkey

Open "Inkey.TXT" For Output As #1
Print #1, "YNM";
Close
Open "Inkey.TXT" For Input As #1
Dim Key As String, Prompt(3) As String
Prompt(1) = "Yes "
Prompt(2) = "No "
Prompt(3) = "Maybe "
Dim Ctr As Integer
Ctr = 0
While Not(EOF(1))
   Key = Inkey(1)
   Ctr = Ctr + 1
   Print Prompt(Ctr); Key
Wend
Close #1
Print "Happy with the results? (YN) ";
Do
  Key = UCase(Inkey)
Loop Until Instr("YN",Key) > 0
```

INPUT Declaration

Open a text or sequential file for the input of data.

Syntax

Open *FileName* For Input As #*FileNum*

Remarks

The **Input** declaration is an integral part of the **Open** statement for opening a file. Use **Input** where the file exists and contains data that is to be read into the program variables using the **Input** #*FileNum* statement. The **Input** declaration is only used for text or sequential files.

See also

Append, Binary, Open, Output, Random

Example

```
Rem Display the contents of a file

Dim tStr As String
Open "Birthday.TXT" For Input As #1
While Not(EOF(1))
    Input #1, tStr
    Print tStr
Wend
Close #1
```

INPUT Statement

Read a line of text from the keyboard or a file.

Syntax

Input [#*FileNum*,] *String*

Input [#*FileNum*,] *Num*

Remarks

The **Input** statement is the basic means of getting keyboard-entered data from the user. The **Input** statement will continue to accumulate keystrokes until terminated with the [Enter] key. The data entered will be stored in the *String* or *Num* variables. The Num variable can be any valid numeric data type of **Byte**, **Integer**, **Long** or **Real**.

When the **Input** statement is called with the optional *FileNum* it will read a line of text data from a file opened 'For Input' with the *FileNum* file variable. The file data will be stored in *String*.

When using **Input** with a file, a 'line of text' is considered as all characters up to and including the next carriage return and line feed characters in the file. When the *String* is read from the file the carriage return and line feed characters are not included as they merely indicate the end of the string.

See also

Get, Inkey, InputMask, InputNumber, InputYesNo, Open, Print, Put

Example

```
Rem Read the keyboard and a file

Dim StrA As String
Print "Enter your access code ";
Input StrA
If UCase(StrA) <> "SYSOP" Then End
```

```
Open "Comments.DAT" For Input As #1 : Rem Assume file exists
While Not(EOF(1))
    Input #1, StrA
    Print StrA
Wend
Close #1
```

INPUTMASK Function

Supply a prompt and read a formatted line of text.

Syntax

InputMask(*Prompt, Mask***)** {As String}

Remarks

The **InputMask** function will present the user with a *Prompt* and a formatted string ready to fill in the blanks. The characters that can be used in a mask string are...

"E"	No character is allowed in this position.
"X"	Any character may be used here including leaving it blank.
"9"	Only numbers will be accepted in this position.
"Y"	Only a "Y" or "N" response is accepted

All other characters in the mask string are considered non-mask characters and will be skipped during the keyboard entry. The masking function is case sensitive, they must be declared in upper case letters.

This function is ideally suited to entering formatted numbers like Social Security, date of birth or credit card numbers.

See also

Inkey, Input, InputNumber, InputYesNo

Example

```
Rem Get several formatted strings from the caller

Dim SSNStr As String*11, PhoneStr As String*14
SSNStr = InputMask("Your Social Security # ","999-99-9999")
PhoneStr = InputMask("Your home phone # ","(999) 999-9999")
Print "Thank you for SSN "; SSNStr; " and "; PhoneStr
```

INPUTNUMBER Function

Read numbers from the keyboard with range checking.

Syntax

InputNumber(*Prompt*, *Low*, *High*) {As Long}

Remarks

The **InputNumber** function provides a direct method of reading the data from the keyboard, while checking it is valid numeric data within a specified range.

The *Prompt* is a string prompt to precede the user entry with *Low* and *High* for checking that the numbers entered are within a prescribed range.

See also

Inkey, Input, InputMask, InputYesNo

Example

```
Rem Get a number in a specified range from the caller
Rem Let Wildcat! take care of the range checking

Dim Answer As Integer, RandNum As Integer
Do
   Answer = InputNumber("Enter a number between 7 and 23 ",7,23)
   RandNum = ((Rnd * 100) Mod 16) + 7
   If RandNum = Answer Then
      Print "Great your time has just been bumped by "; Answer
      AddTime Answer
   ElseIf (RandNum + 1) = Answer Or (RandNum - 1) = Answer Then
      Print "Close, and a little time has been added."
      AddTime Answer \ 2
   Else
```

```
        Print "Ooops not even close. Time reduced"
        AddTime -2
    End If
    Print "Time left online: "; TimeLeft
Loop Until Answer = 23
```

INPUTYESNO Function

Read a yes or no response from the keyboard

Syntax

InputYesNo(*Prompt*) {As True/False}

Remarks

Wait for the user to enter a Y or N response to the question of the *Prompt*. The user will have to press ENTER after the selection to terminate the function and return the **True** (Yes) or **False** (No) response. Anything other than Yes or No will display an error message and return to wait for one or the other.

This function will use the designated **YesChar** and **NoChar** characters to evaluate the caller's response.

See also

Inkey, Input, InputMask, InputNumber, NoChar, YesChar

Example

```
Rem InputYesNo automatically uses YesChar/NoChar


Dim FName As String
Print "Enter the file name to read ";
Input FName
If Exists(FName) Then
  If InputYesNo("Ready to read the text file ") Then
   DisplayTextFile(FName)
  End If
End If
```

INSTR Function

Search for the occurrence of one string within another.

Syntax

Instr([*Start,*] *SearchString*, *SubString*) {As Integer}

Remarks

The **Instr** function is used to locate the existence of a sub-string within another string. If a match is found then the starting position of the sub-string is returned. If no match is found then zero is returned.

Start is an optional position for **Instr** to begin the search from. Normally **Instr** will begin at position number one in the search string however if you want to find all occurrences of the *SubString* then *Start* is used to increment the start position along the string.

See also

Left, Mid, Right

Example

```
Rem Identify several occurrences of a sub-string


Dim fPos As Integer

Const tStr = "The moving finger writes, and having writ moves on"

Const sStr = "VING"

fPos = 0 : Rem We need to start somewhere

Print tStr

Do

  fPos = Instr(fPos + 1, UCase(tStr), sStr)

  if fPos > 0 Then

    Locate CsrLin, fPos
```

```
    Print Mid(tStr,fPos,Len(sStr));
  End If
Loop until fPos = 0
Print
```

INT Function

Return the integer portion of a number.

Syntax

Int(*Val*) {As Long}

Remarks

The **Int** function will remove the fractional or decimal amount from the *Val* numeric value and return just the portion to the left of any decimal point. The Int function can be used to find the remainder from a division as one of the following examples shows.

The **Fix** and **Int** functions operate the same for positive values, however with negative values their operation is slightly different. **Fix** returns the next-higher integer with negative values, while **Int** returns the next lower integer with negative values.

See also

Fix, Mod

Example

```
Rem Fix and Int examples

Dim RealA As Real, IntA As Integer
RealA = 123.456789
IntA = Int(RealA / 13)
Print "RealA divided by 13 = "; IntA;
Print " plus "; Int(RealA) - (IntA * 13); " over"
Print "We could also use Mod "; IntA;
Print " plus "; Int(RealA) Mod 13; " over"
Print
Print "Fix(3.7) = "; Fix(3.7)
Print "Int(3.7) = "; Int(3.7)
Print "Fix(-4.9) = "; Fix(-4.9)
Print "Int(-4.9) = "; Int(-4.9)
Rem This will return 3 and -4 for Fix and 3 and -5 for Int.
```

INTEGER Type

An internal data type for storing small signed values.

Syntax

Dim *Name* As Integer

Remarks

The term 'signed' number simply refers to the sign, either plus or minus that can precede the value indicating it is either a positive or negative number or value.

Variables of **Integer** type can hold signed values in the range -32768 to 32767.

See also

Byte, Dim, Integer, Let, Long, Real, String, Type

Example

```
Rem Rudimentary integer declaration

Dim IntA As Integer
IntA = 123
Print "This integer currently equals "; IntA
```

JOIN Statement

Join an existing conference.

Syntax

Join *Num*

Remarks

Immediately joins the user to the *Num* conference as though the user had selected the [J]. Join conference menu option.

Example

```
Rem Join any valid conference

Dim KeyIn As String*1
Const MenuName = "TECH SUPPORT:"
Print
Print MenuName
Print "[Q]-QModem tech support conference"
Print "[W]-Wildcat! tech support conference"
Print "[X]-Exit back to menu"
Print
Print MenuName; " [Q W X] ? ";
Do
   Input KeyIn
Loop Until Instr("QWX",UCase(KeyIn)) > 0
Select Case KeyIn
   Case "Q", "q"
      Join 54 : Rem This conference must exist
   Case "W", "w"
      Join 27 : Rem This conference must exist
   Case Else
      End
End Select
```

KILL Statement

Kill or delete a file.

Syntax

Kill *FileName*

Remarks

The name *FileName* is the file to **Kill**. This statement is for deleting single, named files only. DOS wildcards * and ? are not supported.

To delete groups of files use **Kill** in conjunction with **FindFirst** and **FindNext** procedures where the DOS wildcard characters *are* supported and the full file name can be passed to the **Kill** statement.

You cannot **Kill** a file that is already open by your *wcCODE* program or hopefully any other program but that responsibility must fall to the operating system to monitor. DOS may let you **Kill** an open file which can often lead to disastrous results. It is therefore your responsibility to ensure that all files about to be killed are under you program's control and have been closed.

Using the **Close** command without any parameters will close *all* open files in use by your program.

The **Del** and **Kill** commands are identical in their operation and can be freely interchanged. The reason we include both is to maintain consistency with the original BASIC language on which *wcCODE* is based. This duplication is to assist those programmers who prefer to use keywords or commands they are more familiar with rather than dictate our own opinions on what *we* think they should use.

See also

Del, RmDir, FindFirst, FindNext, Close

Example

```
Rem Attempt to delete the Test.DAT file

Const FName = "Test.DAT"
If Exists(FName) Then   : Rem First check to see if it is there
   Kill(FName)
Else
   Print "Could not find "; FName
End If
```

LCASE Function

Convert all alphabetic characters in a string to lower case.

Syntax

LCase(*String*) {As String}

Remarks

This function will force all alphabetic characters in any string to lower case. It is the opposite of **UCase** which forces all alphabetic characters to upper case. The characters affected are 'A' through 'Z', 'a' through 'z' and any diacritical-marked characters in the international character-set in the current DOS configuration.

The *String* itself is not affected unless the function result is assigned to it, e.g. *String* = LCase(*String*).

See also

UCase

Example

```
Rem Capitalize a name

Dim NameStr As String
Print "Enter your first name please ";
Input NameStr
NameStr = LCase(NameStr)
NameStr = UCase(Left(NameStr,1)) + Mid(NameStr,2,Len(NameStr)-1)
Print "G'day "; NameStr
```

LEFT Function

Return a specified number of characters from a string.

Syntax

Left(*String*, *Num*) {As String}

Remarks

The **Left** function will copy the number of characters defined by *Num* into another string. If *Num* is greater than the length of the string then the entire string is returned.

The *String* need not be a variable it can just as easily be a literal string of characters as in **Left**("0123456789", 5) would return "01234"

See also

Instr, Mid, Right

Example

```
Rem A Token function, antithesis to one in Right(Str,Num)

Function BeforeChar(TargetCh As String, SearchStr As String) As String
   Dim JunkStr As String
   Rem Add TargetCh to SearchStr just in case there is none
   JunkStr = Left(SearchStr, Instr(SearchStr + \
                 TargetCh, TargetCh)-1)
   BeforeChar = JunkStr
End Function

Const ShortStr = "The Wildcat & other bedside stories"
Print ShortStr
Print BeforeChar("&",ShortStr)
```

LEFTPAD Function

Pad the left side of a string with blanks.

Syntax

LeftPad(*String*, *Len*) {As String}

Remarks

When **LeftPad** is used it will return the *String* padded out to *Len* length by filling in the left side of the string with blank spaces. If the *String* is already equal to or longer than *Len*, then it is returned exactly as is. The *String* itself is not affected unless the function result is assigned to it, e.g. *String* = LeftPad(*String*, *Len*).

The *String* is padded on the left until the total length is equal to *Len* rather than *Len* blanks being added to the left of *String*.

See also

LTrim, Pad, RTrim, Space, Spc, String, Trim

Example

```
Rem Display a menu in uniform columns

Const MenuWid = 25
Function Dots(Prompt As String, Desc As String) As String
   Dim tStr As String*1, tDesc As String
   If Prompt = "" Then
      tStr = UCase(Left(Desc,1))
      tDesc = tStr + Mid(Desc,2,Len(Desc) - 1)
   Else
      tStr = UCase(Prompt)
      tDesc = Desc
   End If
```

```
    Dots = "[" + tStr + "]" + String(MenuWid - \
          Len(tDesc), ".") + tDesc
End Function
Const MenuName = "NEAT MENU:"
Const MenuOffset = MenuWid + 12
Print
Print MenuName
Print
Print Dots("","Be there");
Print LeftPad(Dots("","or be square"), MenuOffset)
Print Dots("","Leaving home");
Print LeftPad(Dots("?","for a far better place"), MenuOffset)
Print
```

LEN Function

Return the number of characters in a string or other data type.

Syntax

Len(*String*) {As Integer}

Len(*TypeDec*) {As Integer}

Remarks

In the first construct **Len** returns the number of characters in the *String* as an Integer value. The **Len** function includes all ASCII characters in the string even if they are unprintable or are blank spaces padding the left or right of the string.

The second construct for **Len** will determine the total length or size of *TypeDec* any data type. See the example below as it is easier to demonstrate than describe this use of the **Len** function. The use of **Len** in the second construct should always be used in the **Open** statement for opening **Random** files. Using **Len** will ensure your data files are always opened with the exact size of the *TypeDec* record.

See also

InStr, Left, Mid, Open, Right, String, Type

Example

```
Rem Len used for strings and user-data types

Const ExactLen = 5
Type UserData
    Name As String*30
    Alias As String*ExactLen
    Age As Integer
End Type
Dim uData As UserData : Rem Assume file exists
```

```
Open "Testdata.DAT" For Random As #1 Len = Len(UserData)
Do
    Print "Your "; ExactLen; " character alias: ";
    Input uData.Alias
Loop Until Len(uData.Alias) = ExactLen
Put #1,1,uData
Close #1
```

LET Statement

Assign the value of an expression to a variable.

Syntax

[Let] *Name* = *Expr*

Remarks

Using **Let** is completely optional when assigning a value *Exp* to a *Name* variable. The variable being assigned must first have been declared using the **Dim** statement and only a value of the correct data type can be assigned.

The **Let** command is a 'carry-over' from the original BASIC language on which *wcCODE* is based.

See also

Dim, Type

Example

```
Rem Let is always implied in assignment

Dim AnInt As Integer, Astring As String

Let AnInt = 32000
Let Astring = "The use of Let is optional"
Print AnInt; ".."; Astring
AnInt = 27
Astring = "This is easier"
Print AnInt; ".."; Astring
```

LISTFILES Statement

List all available files in the files database.

Syntax

ListFiles

Remarks

The ListFiles statement provides the user with the same options as selecting the [L]..List files menu option.

See also

ListFilesDate

Example

```
Rem List the files until told to quit

Function GetKey As String
    Dim tKeyIn As String*1
    Do
        tKeyIn = UCase(InKey)
    Loop Until tKeyIn > ""
    Print tKeyIn;
    GetKey = tKeyIn
End Function

Dim GoHome As Integer, KeyIn As String*1
Do
    Print
    Print "You have selected the Old files section!"
    print
    Print "Press [L].......List files        [Q]..........Quit to menu"
```

```
    Print
    Print "Selection: ";
    KeyIn = GetKey
    Select Case KeyIn
        Case "L"
            Print : ListFiles
        Case "Q"
            GoHome = True
        Case Else
            Beep
    End Select
Loop Until GoHome
```

LISTFILESDATE Statement

List all available files according to date criteria.

Syntax

ListFilesDate

Remarks

This is almost the same as the **ListFiles** statement except that it will list only those files that meet the file-date criteria entered by the user.

See also

ListFiles

Example

```
Rem List the files until told to quit

Function GetKey As String
    Dim tKeyIn As String*1
    Do
        tKeyIn = UCase(InKey)
    Loop Until tKeyIn > ""
    Print tKeyIn; : GetKey = tKeyIn
End Function

Dim GoHome As Integer, KeyIn As String*1
Do
    Print : Print "You have selected the Oldfiles section!"
    Print : Print "[L]..List  [N]..New  [Q]..Quit to menu"
    Print : Print "Selection: "; : KeyIn = GetKey
    Select Case KeyIn
        Case "L"
```

```
        Print : ListFiles
    Case "N"
        Print : ListFilesDate
    Case "Q"
        GoHome = True
    Case Else
        Beep
  End Select
Loop Until GoHome
```

LOC Function

Return the position of the open-file pointer.

Syntax

Loc(*FileNum*) {As Long}

Remarks

The **Loc** function will return the position of the current pointer for an open *FileNum* file but this works differently for the various file types.

For files opened for **Input, Output** or **Append** as text files the **Loc** function will return the value in bytes. Text files use **Print** and **Input** to access the data and the **Loc** function will indicate a value two bytes greater than expected for each line of text as every string has a carriage return (**Chr**(13)) and line feed (**Chr**(10)) added to the end.

For files opened for **Random** as record-structured data files the value returned by **Loc** is the record number within the file. The record number is set by the **Len** = identifier in the **Open** file statement. For a **Random** file the actual byte position can be calculated by multiplying the **Len** = value by the record number returned by **Loc**, however there is probably no reason you might want to do this.

For files opened for **Binary** as individual-byte data files the **Loc** function operates strictly on a byte count to indicate the position of the file pointer.

Note!

The files in *wcCODE* all start at byte or record position one (1) unlike some other program languages which start at byte or record zero (0).

See also

Get, Input, Open, Output, Put, Seek

Example

```
Rem Demonstrate the difference between, LOF and Loc

Const FName = "Birthday.TXT" : Rem Make sure file exists
If Exists(FName) Then
   Open FName For Append As #1
   Print "When opened for 'Append' the Loc and LOF differ by 1"
   Print "Loc = "; Loc(1); " LOF = "; LOF(1)
Else
   Print "Could not find the file"
End If
```

LOCAL Function

Return true if the user is logged on locally.

Syntax

Local {As True/False}

Remarks

This function will return either **True** or **False** if a user is logged onto *Wildcat!* locally rather than via a modem and phone line.

Example

```
Rem Provide access only if the user is logged on locally

Dim KeyIn As String*1
Do
   Print "AUX MENU:" : Print "[S]..Shell to DOS  [Q]..Quit to menu";
   Print : Print "Select [S Q] : "; : Input KeyIn
Loop Until Instr("SQsq",KeyIn) <> 0
Select Case UCase(KeyIn)
   Case "S"
      If Not(Local) Then
         Print "This option is not available when on-line"
      Else
         Shell
      End If
   Case "Q"
      End
End Select
```

LOCATE Statement

Position the cursor on the screen.

Syntax

Locate *Row*, *Column*

Remarks

The **Locate** statement will move the cursor to a position on the screen according to the coordinates supplied by *Row* and *Column*.

The *Row* value is the number of the screen row at which the cursor is to be positioned. This value should be in the range from 1 through to the maximum number of rows on the screen.

The *Column* is the number of the screen column at which the cursor is to be positioned. This value should be in the range from 1 through to the maximum number of columns on the screen, usually 80.

Note!

This command assumes the caller has ANSI capability. If the command **ANSIDetected** returned **False** then **Locate** will not work.

See also

Cls, ClrEOL, CsrLin, Pos

Example

```
Rem Position the cursor, display a prompt as a screen saver

Function ScreenSave As String
    Dim Row As Integer, Col As Integer
    Dim Key As String
    Cls
    Do
        Row = ((Rnd * 100) Mod 24) + 1
```

```
        Col = ((Rnd * 100) Mod 69) + 1
        Locate Row, Col
        Print "Waiting...";
        Delay 0.25
        Locate Row, Col
        Print "          ";
        Key = Inkey
    Loop Until Key > ""
    ScreenSave = Key
End Function
ScreenSave : Rem Main body of program here
```

LOCATEUSER Statement

Display a users details.

Syntax

LocateUser

Remarks

The **LocateUser** statement will request a name of a user and then display details on that user if they exist in the database.

When **LocateUser** is called it returns exactly the same response as if the user had selected the [V]..Verify a user menu option.

Example

```
Rem Identify a user

LocateUser
```

LOF Function

Return the number of records in an open file.

Syntax

LOF(*FileNum*) {As Long}

Remarks

The **LOF** function is an acronym for **L**ength **O**f **F**ile which can have different meanings for different types of files.

If the file was opened for **Input**, **Output**, **Append** or **Binary** access then the **LOF** function will return the size of the file in bytes. If the file was opened for **Random** access then the value returned will be the number of records in the file based on the **Len** = declaration in the **Open** file statement.

See also

EOF, Loc, Open

Example

```
Rem Add another record to the database

Type BadGuyRec
    Name As String*30
    Alias As String*30
    TimesOnBBS As Integer
End Type
Dim BadGuy As BadGuyRec
Open "BadGuy.DAT" For Random As #1 Len = Len(BadGuyRec)
BadGuy.Name = "Remington Blair"
BadGuy.Alias = "NoOne Home"
BadGuy.TimesOnBBS = 3
Put #1, LOF(1) + 1, BadGuy : Rem Position after the last record
Close #1
```

LOG Function

Return the natural logarithm of a numeric expression.

Syntax

Log(*Val*) {As Real}

Remarks

The **Log** function will return the natural Logarithm of the numeric expression *Val* providing it is always positive. An error will be declared if you attempt to use the **Log** function for a negative or zero value. This error can be handled using the **Catch Err_Math**.

See also

Catch, Exp

Example

```
Rem The Log of a number

Print "The Log of 1 is "; Log(1)
```

LONG Type

An internal data type for storing large signed values.

Syntax

Dim *Name* As Long

Remarks

The term 'signed' number simply refers to the sign, either plus or minus that can precede the value indicating it is either a positive or negative number or value.

Variables of the **Long** type can hold signed values in the range -2147483648 to 2147483647.

See also

Byte, Dim, Integer, Real, Type

Example

```
Rem Typical use of a Long data type

Dim Pay As Long
Const HoursPW = 50, Rate = 33, WeeksPY = 52
Pay = HoursPW * Rate * WeeksPY
Print "My annual salary is $"; Pay
```

LOOP Statement

An integral part of the Do...Loop construct.

Syntax

Do [{While | Until} *Condition*]

Statements...

Loop

Do

Statements...

Loop [{While | Until} *Condition*]

Remarks

The **Loop** statement is never used without its partner the **Do** statement therefore the example here is exactly the same as that for **Do**. The function of **Loop** is to indicate the end of the code that forms the section to be repeated.

See also

Do, Exit Do, For...Next, While...Wend

Example

```
Rem Do something with a Do ... Loop

Dim ByteA As Integer
Const WaitingStr = " I'm still here and working hard"
ByteA = 0
Do
   Print ByteA; WaitingStr
   ByteA = ByteA + 1
Loop Until ByteA > 10
```

LTRIM Function

Trim leading spaces from a string.

Syntax

LTrim(*String*) {As String}

Remarks

The value *String* is any valid string type expression which will have all of the leading spaces removed from it. The term 'leading spaces' refers to any spaces to the left of the first non-space character.

In the following example the **LTrim** and **RTrim** functions are used to ensure there are no blanks on either side of the user entered data so that formatting into neat, left justified columns is assured.

See also

RTrim, Trim

Example

```
Rem Strip out all blanks on the left of a string

Dim tStr(6) As String, Ctr As Integer
tStr(1) = "    A little ditty     "
tStr(2) = "  hello Dolly     "
tStr(3) = "I said hello Dolly"
tStr(4) = "It's been good to see        "
tStr(5) = "     you way back when       "
tStr(6) = "        err I don't know any more words"
For Ctr = 1 To 6 Step 2
  Print Pad(LTrim(RTrim(tStr(Ctr))),40);
  Print LTrim(RTrim(tStr(Ctr + 1)))
Next Ctr
```

MAXNODE Function

Return the maximum nodes for the installed version of *Wildcat!*.

Syntax

MaxNode {As Integer}

Remarks

The **MaxNode** command will return the maximum number of nodes the currently running version of *Wildcat!* is capable of supporting. This value will be in the range of one to **MaxNode**.

The value returned will always be one-less than the actual number of nodes available, as node zero (local logon) is a valid node number on all *Wildcat!* systems.

Example

```
Rem All Wildcat! versions support at least two nodes

Print "The maximum this version can support is "; MaxNode
Print "and don't forget Node zero!"
```

MID Function

Retrieve a sub-string from within another string.

Syntax

Mid(*String*, *StartPos*, *Count*) {As String}

Remarks

The **Mid** function will allow you to copy a designated section of the *String* by indicating the *StartPos* as the start position and the *Count* as the number of characters to copy.

If the length of *String* is less than *StartPos* + *Count* then all that will be returned is the number of characters up to the end of *String*.

See also

Left, Right

Example

```
Rem Extensive demo using in-part, the Mid function

Function GetMins(Prompt As String) As Long
  Dim tStr As String, Hours As Long, Mins As Long
  tStr = InputMask(Prompt, "99:99")
  Hours = Val(Left(tStr, 2))
  Mins = Val(Mid(tStr, 4, 2))
  GetMins = (Hours * 60) + Mins
End Function

Dim StartM As Long, EndM As Long
Dim WorkH As Long, WorkM As Long
Do
  Print "Enter zero time to quit"
```

```
  StartM = GetMins("Start time ")
  If StartM <> 0 Then
    EndM = GetMins("End time   ")
    WorkM = EndM - StartM
    WorkH = WorkM \ 60 : Rem Note the backslash for integer division
    WorkM = WorkM - (WorkH * 60)
    Print "hours worked = "; WorkH; ":"; WorkM
  End If
Loop Until StartM = 0
```

MKDIR Statement

Make a new directory on the disk.

Syntax

MkDir *Directory*

Remarks

Make a new directory with the name *Directory* in the current directory. If a full path and new directory name is supplied then all directories leading up to the new one *must* already exist. It is not possible to make more than one directory at a time.

If an attempt is made to make a directory on a path that does not yet exist then the Err_Path error will be declared. This error can be caught by using the Catch command and the Err_Path identifier.

Note!

Having made a new directory, please note that the *wcCODE* program language has no means of changing out of the current one. Consequently any use of the newly-made directory must be accessed by using its full path name.

See also

RmDir

Example

```
Rem Make a new directory for file storage

Function NewDirOk(tStr As String) As Integer
   Rem Error Err_Path : Rem Used for testing Catch
   NewDirOk = True
   MkDir(tStr)
   Catch Err_Path
      NewDirOk = False
      Print "Unable to make "; UCase(tStr)
      WaitEnter
End Function

If NewDirOk("C:\Wildcat\work\Data3") Then
   Print "New path created"
End If
```

MOD Operator

Return the remainder of a division between two values.

Syntax

Op1 Mod *Op2*

Remarks

The **Mod** operator will return the remainder of an integer division between the two operators *Op1* as the numerator and *Op2* as the denominator.

Op1 and *Op2* are any integer numeric type of **Byte**, **Integer** or **Long**. If *Op2* is zero then the **Err_Math** error will be generated.

Example

```
Rem Use Mod to provide numbers between 1 and 24, 1 and 69


Function ScreenSave As String
   Dim Row As Integer, Col As Integer
   Dim Key As String
   Cls
   Do
      Row = ((Rnd * 100) Mod 24) + 1
      Col = ((Rnd * 100) Mod 69) + 1
      Locate Row, Col
      Print "Waiting...";
      Delay 0.25
      Locate Row, Col
      Print "          ";
      Key = Inkey
```

```
    Loop Until Key > ""
    ScreenSave = Key
End Function
ScreenSave : Rem Main body of program here
```

MOREPROMPT Function

Enable, disable or report the status of the Pause prompt.

Syntax

MorePrompt [*On* | *Off*] {As True/False}

Remarks

This command when called with the optional parameters, will enable or disable *Wildcat!*'s "Pause" prompt. If it is called without parameters it will return the current status of the "Prompt". This way your program can check to see if the "Prompt" is being displayed or not. When displaying large quantities of text it is good practice to display the prompt occasionally just give the caller the opportunity to abort the process should they so desire.

See Also

DisplayStopped

Example

```
Rem Assume the file is a long text file

Const MaxCtr = 200
Dim Ctr As Integer, KeepGoing As Integer
Dim tStr As String
Open "LongText.TXT" For Input As #1 : Rem Assume it exists
KeepGoing = InputYesNo("Display large lumps at a time? ")
If KeepGoing Then MorePrompt Off
While Not(EOF(1))
   Ctr = Ctr + 1
   Input #1, tStr : Print tStr
      Rem Check every so often if large lumps displayed
   If (KeepGoing) And (Ctr > MaxCtr) Then
      Ctr = 0
```

```
        Pause
        If DisplayStopped Then GoTo HadEnough
    End If
Wend
HadEnough:
Close #1
```

MOVELEFT Statement

Move the cursor left.

Syntax

MoveLeft *Num*

Remarks

The **MoveLeft** statement as the name implies, moves the cursor to the left the *Num* number of spaces.

See also

CsrLin, Locate, MoveRight, Pos

Example

```
Rem Position the cursor under the [Enter] prompt

Function PressEnter As Integer
   Const Prompt = "Press [Enter] to proceed"
   Const OkChars = Chr(13) + Chr(27)
      Rem Enter is ASCII 13 and Esc is ASCII 27
   Dim KeyIn As String*1, Ok As Integer
   Print Prompt;
   MoveLeft Len(Prompt) - Instr(Prompt,"[")
   Do
      KeyIn = Inkey
      Ok = Instr(OkChars, KeyIn) <> 0
      If Not Ok Then Beep
   Loop Until Ok
   PressEnter = Asc(KeyIn)
End Function
If PressEnter = 13 Then
```

```
    Print
    Print "[Enter] Now you're talkin!"
Else
    Print
    Print "[Esc] No escape from Alcatraz Harry!"
End If
```

MOVERIGHT Statement

Move the cursor right.

Syntax

MoveRight *Num*

Remarks

The **MoveRight** statement as the name implies, moves the cursor to the right the *Num* number of spaces.

See also

CsrLin, Locate, MoveRight, Pos

Example

```
Rem The basis for a line editor.

Dim KeyIn As String, Col As Byte
Dim WorkStr As String, tStr As String
Const LeftArr = Chr(75), RightArr = Chr(77)
Const ExitChars = Chr(13) + Chr(27)
Col = 1 : Print : tStr = "My nifty little editor"
Print tStr; : Locate CsrLin, Col : WorkStr = tStr
Do
   Do
      KeyIn = Inkey
   Loop Until KeyIn > ""
   If KeyIn = Chr(0) Then
      KeyIn = Inkey
      Select Case KeyIn
         Case LeftArr
            If Col > 1 Then
               Col = Col - 1
               MoveLeft 1
            Else
               Beep
            End If
```

```
        Case RightArr
            If Col < 80 Then
                Col = Col + 1
                MoveRight 1
            Else
                Beep
            End If
    End Select
Else
    Select Case KeyIn
        Case " " To "|"
            Print KeyIn;
            tStr = Left(tStr, Col - 1) + KeyIn + \
                   Right(tStr, Len(tStr) - Col)
            Col = Col + 1
        Case Chr(27) : Rem The Esc key so put string back
            tStr = WorkStr
        Case Chr(13) : Rem The Enter key
        Case Else
            Beep
    End Select
End If
Loop Until Instr(ExitChars, KeyIn) > 0
Print
Print tStr
```

NAME Statement

Rename or move a file.

Syntax

Name *OldFileName* **As** *NewFileName*

Remarks

The **Name** command will rename a file from the *OldFileName* to the *NewFileName* if only the file name is used. If a complete path name is used for the *NewFileName*, then **Name** will actually move the *OldFilename* file to the *NewFileName* location.

The **Name** command can only be used to rename or move files and cannot be used to rename a directory. In the "move" mode, the files can only be moved to the current drive. You will need to use **CopyFile** to achieve a move to another drive.

The "move" mode actually modifies the file's directory entry rather than physically copying and deleting. The resultant move is far quicker as it involves no data transfers.

If an error occurs with the use of the **Name** command then the **Err_FileRename** error will be returned which can be handled with the **Catch** command.

See also

Del, FindFirst, FindNext, Kill

Example

```
Rem Rename or Move a file

Dim BakFName As String, Today As Date
Dim NewFName As String, KeyIn As String
CurrentDate Today
BakFName = FormatDate(Today, "mm-dd") + "Usr.LOG"
```

```
Name "User.LOG" As BakFName
If Exists("Backup\" + BakFName) Then
  Print "File exists. [R]..Rename  [D]..Delete  [Q]..Quit ";
  Input KeyIn
  Select Case KeyIn
    Case "Q", "q"
      End
    Case "D", "d"
      Del(BakFName)
    Case "R", "r"
      Print "Old name = "; BakFName; " New name ";
      Input NewFName
      Name "Backup\" + BakFName As "Backup\" + NewFName
  End Select
End If
Name BakFName As "Backup\" + BakFName
```

NEWSLETTER Statement

Display the Newsletter

Syntax

NewsLetter

Remarks

This command is exactly the same as selecting the [N]..Newsletter menu option.

For the **NewsLetter** command to work there must be an existing newsletter named NEWSLTR.BBS in the current conference display directory. If none exists then **NewsLetter** will carry on without error or notification to the caller.

Example

```
Rem Display the Newsletter to an online caller

If Not(Local) Then
   NewsLetter
End If
```

NEXT Statement

An integral part of the For...Next loop procedure.

Syntax

For *Index* = *StartVal* **To** *EndVal* [**Step** *StepVal*]

Statements...

Next [*Index*]

Remarks

The *Index* is a numeric variable of type **Byte**, **Integer**, **Long**, or **Real** which is incremented each time the loop is performed. The *StartVal* value is the starting point or counter of the loop. The *Index* is set to this value at the start of the **For**...**Next** sequence. The *EndVal* value is the maximum value for the *Index* to reach for the **For**...**Next** process to terminate.

The *Step* is an optional designator that defines the amount by which the counter is incremented after each pass in a **For**...**Next** sequence. When the *Step* value is omitted the increment defaults to one. The *Step* value can also be a negative number to allow a decrement downwards.

The *Index* following the **Next** statement is optional but for programming clarity it is a good idea to include it. This is especially true when you have several nested **For**...**Next** loops all ending with a series of **Next Next Next** statements. Do yourself a favor and prepare for easy debugging when needed (and not **if** it's needed).

See also

For, Do...Loop, Exit For, While...Wend

Example

```
Rem Display a count-down timer

Dim Ctr As Integer, Row As Integer, Col As Integer
Print "Waiting for keyboard response : ";
Row = CsrLin
Col = Pos
For Ctr = 30 To 1 Step -1
   Locate Row, Col
   Print Ctr; " "; : Rem Stomp on extra digit when we get below 10
   Delay 1
   If InKey > "" Then
      Exit For
   End If
Next Ctr
```

NOCHAR Function

Return the local language character for the word No.

Syntax

NoChar {As String}

Remarks

This function will return the equivalent local language character to the letter 'N' as the first character in the English word 'No'. This function is intended for evaluating Yes/No responses in your programs.

The **NoChar** and **YesChar** values are used by the **InputYesNo** function to provide multi-lingual support. They can also be used by your programs to determine the caller's response without having to know if they have elected to use foreign language prompts in *Wildcat!*.

See also

InputYesNo, YesChar

Example

```
Rem Use YesChar and NoChar to test a response

Dim KeyIn As String*1
Print "Are you sure you want to do this? ";
Do
   KeyIn = UCase(Inkey)
Loop Until (KeyIn = YesChar) Or (KeyIn = NoChar)
```

NODE Function

Return the current node number.

Syntax

Node {As Integer}

Remarks

This command will simply return the current node number of the user as an integer in the range 0 through to **MaxNode**.

See also

MaxNode

Example

```
Rem Simple example of the Node number function

Print "Your are "; User.Name; " connected to "; Node
```

NOT Operator

Perform a logical Not operation.

Syntax

Not *Op*

Remarks

This operator takes the binary representation of *Op* and reverses the status of each bit in the representation. When using **Not** in an **If...Then** statement the value of **False** (zero) will be converted to a value of **True** (–1) and vice versa.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

And, Eqv, If, Imp, Or, XOr

Example

```
Rem A comparison using the Not operator

If Not(Exists("Error.LOG")) Then
   Print "Error.LOG seems to be missing!"
Else
   Print "Everything appears in order here."
End If
```

NUMBER Variable field

Storage area for the value of a Date or Time.

Syntax

.Number

Remarks

The **.Number** variable field stores the date or time value in the **Date**, **Time** or **DateTime** variables. As **.Number** is a variable-field it can also have values assigned to it by your program.

See also

CurrentDate, CurrentDateTime, CurrentTime, FormatDate, FormatTime

Example

```
Rem Number is an integral part of Date variables

Dim DateVar As Date
CurrentDate(DateVar)
Print "The date value: "; DateVar.Number; \
    " is really "; FormatDate(DateVar, "dd/mm/yyyy")
```

OCT Function

Convert a value to its octal representation.

Syntax

Oct(*Val*) {As String}

Remarks

The *Val* is the numeric variable or expression that **Oct** function converts into a **String** type. This value may be any numeric type which will be converted to either **Integer** or **Long** representation as appropriate with any fractional values ignored.

See also

Hex, Str, Val

Example

```
Rem Display an octal-base value

Dim OctStr As String
OctStr = Oct(90210)
Print "Oldies but goodies = "; OctStr
Print "The Oct value of 123 is "; Oct(123)
```

OPEN Statement

Open a file for data access.

Syntax

Open *FileName* **For** *AccessMode* **As** [**#**]*FileNum* [**Len** = *RecLen*]

Remarks

The **Open** statement is used to open files for input and output for the various file types supported by *wcCODE*. Each file type has its own syntax variations and no checking is done on the type of file or the data it may contain. It is therefore possible to open a text file for a **Random** search with somewhat meaningless results for the data returned.

The *FileName* is enclosed in quotes or can be a string assigned with the name of the file to use. The *FileNum* can also be assigned to a numeric variable with the pound sign (#) omitted of course. The pound sign is optional in the opening statement but is mandatory with **Get**, **Input**, **Print**, **Put**, **Output** when used to access file data.

The various syntax for each type of *AccessMode*...

- **Open** *Filename* **For Input As #***FileNum*
- **Open** *FileName* **For Output As #***FileNum*
- **Open** *FileName* **For Append As #***FileNum*

Text files are made up of strings with carriage return and line feed characters added as they are written to the file. Text files are often called sequential files as the data is always added sequentially.

The **Output** *AccessMode* will create a new file whether it exists or not *and* without warning that it might be about to overwrite an existing file.

The **Append** *AccessMode* will create a new file if one does not exist or prepare the existing file by positioning the file pointer at the end.

The **Input** *AccessMode* will open the file to input **String** or text data by using the **Input** statement.

- Open *FileName* For Binary As #*FileNum*

  The **Binary** files are files where the record length is one byte and as such are accessed either randomly or sequentially in one byte increments. The **Binary** file is very similar to a **Random** file but with a record length of one byte. The byte information can be any value in the range 0 through 255.

- Open *FileName* For Random As #*FileNum* Len = *RecLen*

  With **Random** access files the *RecLen* record length is the size of each designated record. This is almost always the size of the data type declared in your program. It is also customary, and easier to use the **Len(***Type***)** statement to report the actual size rather than you trying to calculate it.

A word of caution here is that the **Random** file's data must always be read by using the **Get** statement with the *exact* same record structure that was used to **Put** the data there in the first place.

See also

Close, FreeFile, Get, Input, Len, Output, Put, Seek, Type

Example

```
Rem A Little opening and closing going on

Open "Error.LOG" For Append As #1
Print #1, "I should not mess with this file!"
Close #1
Type MyUserRecord
   Name As String*30
   Alias As String*30
   Age As Integer
End Type
Dim MyUser As MyUserRecord
```

```
Open "MyUser.DAT" For Random As #1 Len= Len(MyUserRecord)
MyUser.Name = "Slick Harry"
MyUser.Alias = "Everything's for sale"
MyUser.Age = 29
Put #1, 1, MyUser
Close #1
```

OR Operator

Perform a logical Or operation between operands.

Syntax

Op1 Or *Op2*

Remarks

The Or operator when used in an If...Then statement will return true if either one of the sets of operands equates to true.

The *Op1* and Op2 operands are logical expressions or numeric values of any valid data type of Byte, Integer or long.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

And, Eqv, Imp, Not, XOr

Example

```
Rem Logical and bitwise manipulations with Or


If (12 > 13) Or (10 < 21) Then
   Print "At least one must be true!"
End If
Print "But 10 ORed with 21 = "; 10 Or 21
```

OUTPUT Declaration

Open a text or sequential file for the output of data.

Syntax

Open *FileName* For Output As #*FileNum*

Remarks

The **Output** declaration is an integral part of the **Open** statement for opening a file. Use **Output** where the file is to be created whether one of the same name already exists or not. No warning is given that a current file of the same name exists prior to overwriting. The **Output** statement is only used for text or sequential files.

See also

Append, Binary, Input, Open, Random

Example

```
Rem Overwrite a file after opening for output

If Exists("Birthday.XXX") Then
   If Not InputYesNo("The file exists, overwrite it? ) Then
      Print "No message or file was written!"
      End
   End If
End If
Open "Birthday.XXX" For Output As #1
Print #1, "Happy birthday"
Close #1
```

PAD Function

Pad a string with spaces on the right.

Syntax

Pad(*String*, *Count*) {As String}

Remarks

The **Pad** function will fill out a string to the designated length of *Count* by placing blank spaces to the right of the last character in the string. The *String* itself is not affected unless the function result is assigned to it, e.g. *String* = **Pad**(*String*, *Len*).

This function does *not* place the *Count* number of blank spaces on the end of *String* but rather makes the finished string-length equal to *Count*.

See also

LeftPad, LTrim, RTrim, Trim

Example

```
Rem Using Pad to align data in columns

Dim NameStr(5) As String, Age(5) As Integer, Ctr As Integer
NameStr(1) = "Flair Patterson" : Age(1) = 29
NameStr(2) = "Stephen Queen" : Age(2) = 43
NameStr(3) = "Cindy Cisero" : Age(3) = 55
NameStr(4) = "Talbragard Shrimpton" : Age(4) = 12
NameStr(5) = "Derek Swineborne" : Age(5) = 23
For Ctr = 1 to 5
    Print Pad(NameStr(Ctr),32); Age(Ctr)
Next Ctr
```

PAGESYSOP Statement

Page the Sysop.

Syntax

PageSysop

Remarks

This presents exactly the same result as if the user of your program had instead selected the [P]..Page the Sysop menu option.

Example

```
Rem Offer a caller the chance of chatting with the Sysop

Dim KeyIn As String*1
Do
   Print "Press [P]-Page the Sysop  [Q]-Quit back to the menu (PQ) ";
   Do
      KeyIn = UCase(Inkey)
   Loop Until KeyIn > ""
   Select Case KeyIn
      Case "P"
         PageSysop
      Case "Q"
         Locate CsrLin, 1
         ClrEOL : Print "Bye for now"
         End
      Case Else
         Print
         Print "Invalid response "; KeyIn
         Beep
   End Select
Loop : Rem Condition-less Do...Loop is handled by "Q" selection
```

PAGEUSER Statement

Page an on-line user.

Syntax

PageUser

Remarks

The PageUser statement will send up to four lines of text to a specific user. This command will ask for the name of the user and then present the editor for a four line message to be entered and transmitted.

See also

PageSysop, SendPage

Example

```
Rem Send a four-line notice to a caller

PageUser
```

PAUSE Statement

Suspend execution until the caller responds.

Syntax

Pause

Remarks

The **Pause** command will present a prompt to the user and halt execution until a suitable response has been made to the displayed prompt.

See also

Delay, WaitEnter

Example

```
Rem Assume the file is a long text file and Pause the display

Const MaxCtr = 200
Dim Ctr As Integer, KeepGoing As Integer
Dim tStr As String
Open "LongText.TXT" For Input As #1 : Rem Assume it exists
KeepGoing = InputYesNo("Display large lumps at a time? ")
If KeepGoing Then MorePrompt Off
While Not(EOF(1))
   Ctr = Ctr + 1
   Input #1, tStr : Print tStr
      Rem Check every so often if large lumps displayed
   If (KeepGoing) And (Ctr > MaxCtr) Then
      Ctr = 0
      Pause
```

```
    If DisplayStopped Then GoTo HadEnough
  End If
Wend
HadEnough:
Close #1
```

POS Function

Return the column position of the cursor.

Syntax

Pos {As Integer}

Remarks

The value returned by this function will range from 1 to the maximum number of columns on the screen, usually 80.

See also

CsrLin, Locate

Example

```
Rem Use Pos to control the cursor position

Dim Ctr As Integer, Row As Integer, Col As Integer
Ctr = 20
Print "Select [M]..Menu  [G]..Goodbye"
Print "You have "; Ctr; " seconds to respond ";
Row = CsrLin
Col = Pos
Do
   Locate Row, Col
   Print Ctr; " "; : Rem Stomp on extra characters below 10
   Ctr = Ctr -1
   Delay 1
   Select Case UCase(Inkey)
      Case "M"
         End
      Case "G"
```

```
        GoodBye
    End Select
Loop Until Ctr < 1
HangUp
```

PRINT Statement

Output data to the screen or a file.

Syntax

Print [#*FileNum*,][*String*[[;|,]*String* ...]][;|,]

Remarks

The **Print** statement is used to output data to either the screen or a file. The *Exp* is an expression of any type except a user defined type record but it can be any of the individually defined fields within the user defined record.

Multiple items can be separated with semicolons or commas to affect the output of the display or file structure.

If a semicolon is used then the next **Print** statement will follow immediately on from the end of the last character printed or sent to a file. No carriage return and line feed will be added to the file, so a subsequent **Input** from the file will not see this as a separate item but as part of the previous **Print;** statement.

If a comma is used then the following print statement is positioned at the next 14-character-wide print-zone boundary. This means that a 10 character wide string when printed to the screen with a trailing comma will force the next **Print** *Exp* to start at column 14, leaving a gap of 4 blank spaces. This is primarily intended to make it easier to print numeric data in columns.

The **Print** statement with no *Exp* will print a blank line to the screen or send just a carriage return and a line feed characters to a file which is what is actually happening on the screen also.

See also

CsrLin, Locate, Pos, Tab

Example

```
Rem Use Print to display data on the screen

Dim Dt(5) As String*6, Ctr As Integer
Dim DL(5) As Integer, UL(5) As Integer
Rem Pretend we gathered these stats
Dt(1) = "Jun-10" : DL(1) = 605  : UL(1) = 45
Dt(2) = "Jun-11" : DL(2) = 1204 : UL(2) = 602
Dt(3) = "Jun-12" : DL(3) = 306  : UL(3) = 27
Dt(4) = "Jun-13" : DL(4) = 880  : UL(4) = 88
Dt(5) = "Jun-14" : DL(5) = 1046 : UL(5) = 45
Print "Date", "D/L", "U/L",,"Ratio"
Print String(61,"=")
For Ctr = 1 To 5
   Print Dt(Ctr), DL(Ctr), UL(Ctr),,
   Print Int(UL(Ctr) / DL(Ctr)*100); "%"
Next
```

PROGNAME Function

Return the name of the currently running program.

Syntax

ProgName {As String}

Remarks

This function will return the name of the currently running program in a string type format.

See also

ProgPath

Example

```
Rem Track use of programs. This could be an '$INCLUDE file

Dim DT As DateTime
CurrentDateTime DT
Open "MyUser.DAT" For Append As #1
If LOF(1) = 0 Then
   Print #1, "New user log file: ";
   Print #1, FormatDate(DT.D, "mm/dd/yyyy")
   Print #1, String(34,"*")
   Print #1,
End If
Print #1, ProgName;
Print #1, " "; FormatDate(DT.D, "mm/dd/yy");
Print #1, " "; FormatTime(DT.T, "hh:mm")
Print #1, " "; User.Name;
Print #1, String(34, "=")
Close
```

PROGPATH Function

Return path the of the currently running program.

Syntax

ProgPath {As String}

Remarks

This command will return the full drive and path description including a trailing backslash. While it is not possible to change directories in *Wildcat!* via the *wcCODE* language it is still possible to **Chain** to compiled *wcCODE* programs in another directory.

See also

ProgName

Example

```
Rem Store all of your programs in the same directory

Const FName = "MsgBase.WCX"
If InputYesNo("Care to branch to the message base? ") Then
  If Exists (ProgPath + FName) Then
      Chain ProgPath + FName
   End If
End If
```

PUSHCOMMAND Statement

Load the keyboard buffer with keystrokes.

Syntax

PushCommand *Command*

Remarks

The **PushCommand** will push a *Command* string of characters into the keyboard buffer for *Wildcat!* to process when the program terminates. The maximum length of the string is limited to 20 characters.

Example

```
Rem On exiting a program, read all unread messages

Rem The bulk of your program in here
Rem ...
Rem ...
PushCommand "M R U A"
```

PUT Statement

Write data to Random or Binary files.

Syntax

Put [#]*FileNum*, [*FilePos*], *Variable*

The *FileNum* is the file number assigned to the already open file which was opened *only* as **Random** or **Binary** as an *AccessMode*.

The *Position* value is the **Random** access file record, or **Binary** file byte where the **Put** statement is about to place the data. Note that unlike some other languages, the first record or byte in the file is number 1, not number 0.

If the *Position* is omitted by using a comma as a place-holder then the **Put** statement will place the data wherever the current file pointer is pointing as indicated by the **Loc** function..

The *Variable* is the name of the variable that contains the data to place in the file with the **Put** statement.

See also

Close, Get, Input, Open, Output, Seek, Type

Example

```
Rem Gather data and store in a file until told to quit

Type TempRecord
   Name As String*25
   Password As String*14
   Phone As String*15
   SecLevel As String*10
End Type
Dim tRec As TempRecord, ForceUpper As Integer
Dim AllDone As Integer
```

```
Const Quit1 = "QUIT-EXIT-END-DONE-FINISH-DUTCH?-"
Const Quit2 = "ZERO-SQUAT-NIX-NONE-ZIP-DOODLEY-"
Sub ClearData(Row As Byte, Col As Byte)
    Locate Row, Col
    ClrEOL
End Sub
Function InputStr(Row As Byte, Col As Byte, \
                  Prompt As String) As String
    Dim tStr As String
    ClearData(Row, Col)
    Prompt = Trim(Prompt)
    If Right(Prompt, 1) <> ":" Then
        Prompt = Prompt + ":"
    End If
    Prompt = Pad(Prompt, Len(Prompt) + 1)
    Locate Row, Col
    Print Prompt;
    Input tStr
    If ForceUpper Then
        tStr = UCase(tStr)
        Locate Row, Col + Len(Prompt)
        Print tStr
    End If
    InputStr = tStr
End Function

Rem Start of main program body
Cls
Locate 22, 1
```

```
Print "To finish, enter an exit-word for the Name"
Print Quit1; Quit2;
Open "MyUsers.DAT" For Random As #1 Len = Len(TempRecord)
ForceUpper = True
Do
    tRec.Name = InputStr(1, 13, "Name:")
    AllDone = Instr(Quit1 + Quit2, tRec.Name) <> 0
    If Not(AllDone) Then
        tRec.Password = InputStr(3, 9, "Password:")
        Locate 5, 10
        tRec.Phone = InputMask("  Phone: ","999-999-9999")
        tRec.SecLevel = InputStr(7, 9, "Security:")
        Put #1,,tRec  : Rem The two commas force a sequential Put
    End If
Loop Until AllDone
Close
Cls
```

QUESTIONNAIRE Statement

Run a specified questionnaire.

Syntax

Questionnaire *Num*

Remarks

This command will run the *Num* number questionnaire in the same fashion as if the user had selected it from the questionnaire menu.

If the *Num* questionnaire does not exist then an error message will be returned and the system will continue.

See also

QuestionnaireMenu

Example

```
Rem Get a questionnaire for completion

Dim QNum As Integer
Print "Questionnaires:"
Print
Print "[1]..Sysop quiz  [2]..For sale  [3]..Quit"
Print
Print "Number: ";
Input QNum
If QNum = 3 Then End
Questionnaire QNum
```

QUESTIONNAIREMENU Statement

Display the questionnaire menu.

Syntax

QuestionnaireMenu

Remarks

This command will display the questionnaire menu if one is available, and if not then it will display an error message. The caller is given the list of available questionnaires or the option of quitting back to the *Wildcat!* menu that launched the program. If the caller selects a questionnaire to answer they are returned to the Questionnaire menu until quitting.

See also

Questionnaire

Example

```
Rem Get a questionnaire for completion

QuestionnaireMenu
```

QUOTEOFTHEDAY Statement

Display a quote from the Quote-of-the-day file

Syntax

QuoteOfTheDay

Remarks

Select and display the next quote from the QUOTES.BBS file if it exists in the DISPLAY directory. This will select the next available quote in the file or if at the end of the file it will begin again at number one.

Example

```
Rem Use YesChar/NoChar, then display the next quote

If InputYesNo("Care for a random thought? ") Then
   QuoteOfTheDay
End If
```

RANDOM Declaration

Open a file for random access.

Syntax

Open *FileName* For Random As #*FileNum*

Remarks

The **Random** declaration is an integral part of the **Open** statement for opening a file as a randomly accessed, fixed-record-length file. The use of **Random** will create a file if it does not exist. Use **Random** where the file is to be accessed randomly using **Seek**, **Get** and **Put** for the data transfer. The data contained in a **Random** file can be of any user-defined data type including any of *Wildcat!*'s internal data type structures.

See also

Append, Binary, Input, Open, Output

Example

```
Rem Retrieve data from a file opened for Random access

Type UserRec
   Name As String*30
   Alias As String*30
   LastDate As String*8
   TimesCalled As Integer
End Type
Dim DataF As Integer, URec As UserRec
DataF = FreeFile          : Rem Assume the file exists
Open "Records.DAT" For Random As #DataF Len = Len(UserRec)
While Not(EOF(DataF))
   Get #DataF,, Urec    : Rem Step sequentially from start to end
   Print LeftPad(URec.Name,32); LeftPad(URec.Alias,32);
```

```
    Print LeftPad(URec.LastDate,10); URec.TimesCalled
Wend
Close #DataF
```

READMESSAGES Statement

Read messages in the current conference.

Syntax

ReadMessages

Remarks

This command presents the user with exactly the same response as the [R]..Read messages menu option.

Example

```
Rem If the caller desires, then read the messages

If InputYesNo("Read messages now? (YN) ") Then ReadMessages
```

READTEXTFILE Statement

Read the contents of a text file.

Syntax

ReadTextFile

Remarks

This statement presents the user with exactly the same response as selecting the [R]..Read file menu option. The caller will be presented with a prompt for the file name to read. If the field is left blank then **ReadTextFile** will terminate without error.

See also

ViewFile

Example

```
Rem Use ViewFile and DisplayTextFile to show file contents

Dim KeyIn As String*1
Do
   Print "Inspect file contents..."
   Print "[V]..View archived  [R]..Text files  [Q]..Quit (VRQ) ";
   Input KeyIn
   Select Case UCase(KeyIn)
      Case "V"
         ViewFile
      Case "R"
         ReadTextFile
   End Select
Loop Until (KeyIn = "Q") Or (KeyIn = "q")
```

REAL Type

An internal data type for storing signed floating point values.

Syntax

Dim *Name* As Real

Remarks

The term 'signed' number simply refers to the sign, either plus or minus that can precede the value indicating it is either a positive or negative number or value.

The term 'floating point' refers to the fact that the **Real** type can store numbers with fractional or decimal amounts following a floating decimal point.

Variables of the **Real** type can hold signed floating point values in the range -3.4e38 to 3.4e38.

See also

Byte, Dim, Integer, Long, String, Type

Example

```
Rem A Typical use of the Real data type
Dim PayToday As Real, Rate As Real, Hours As Real
Rate = 17.50
Hours = 10.63
PayToday = Rate * Hours
Print "Today I earned $"; PayToday
```

REM Statement

Include a remark or comment.

Syntax

Rem *Comment*

' *Comment*

Remarks

The **Rem** keyword will cause the compiler to ignore everything following it on that line of text. An alternate to the **Rem** statement is the apostrophe character (') which can be freely interchanged with the **Rem** statement. The reverse however is not true, as the **Rem** statement, when used following any valid program code must be preceded by a colon to first, indicate the end of the program statement.

If the **Rem** statement is the first non-blank character on the line then the colon is not used as there are no programming statements to terminate.

The use of the **Rem** does not make any difference to the compiling or execution speed or size of your compiled programs.

Note!

It is good programming practice to use **Rem** statements liberally to document your code for later revision. Many programmers tend not to sufficiently document their source code in the belief that they will remember exactly what's what when the time comes, but unfortunately this is almost never the case.

Example

```
Rem Different uses of the Comment

Dim KeyIn As String*1        : Rem This comment needs a colon
Print "Press [Y]es or [N]o ";
Do                           ' and this one does not
```

```
    KeyIn = UCase(Inkey)
Loop Until KeyIn > ""
If Not(Instr("YN", KeyIn)) Then
    Beep                        : Rem Keystroke not acceptable
Else
    Print "Thanks"              : Rem We are on our way again
End If
```

RESET Statement

Close all open files.

Syntax

Reset

Remarks

The **Reset** statement works exactly the same as the **Close** statement does when called *without* a file number in that it closes all open files.

See also

Close, End

Example

```
Rem Reset will close all open files

Dim tStr As String
Open "TestIN.DAT" For Input As #1
Open "TestOUT.DAT" For Output As #2
While Not(EOF(1))
   Input #1, tStr
   Print #1, tStr
Wend
Reset : Rem Slam them all closed
```

RETURN Statement

Return from a subroutine.

Syntax

Return

Remarks

The **Return** statement is an integral part of the **GoSub** statement. This will cause the program flow to return to the statement immediately following the branching **GoSub** statement.

See also

GoSub

Example

```
Rem Branching forward/backwards with the GoSub/GoTo

Print "I should always use Sub and Function for branching"
GoSub BadMove
Print "Progam now terminating with the End statement"
End
Rem End of the main program body

BadMove:
   Dim YesSir As String*1
   Print "Do you want to see more of this ";
   Input YesSir
   If UCase(YesSir) = "N" Then
      Return Else GoTo BadMove
   End If
Rem No Return needed here as the If statement takes care of it
```

RIGHT Function

Return a specified number of characters from a string.

Syntax

Right(*String*, *Num*) {As String}

Remarks

The **Right** function will copy the number of characters defined by *Num* from the right hand side of the *String*. The count begins with the right-most character and counts to the left. If *Num* is greater than the length of *String* then **Right** will return the same *String*.

See also

Instr, Left, Mid

Example

```
Rem Handy function, antithesis to one in Left(Str,Num)

Function AfterChar(TargetCh As String, SearchStr As String) As String
   Dim JunkStr As String
      Rem Add TargetCh to SearchStr just in case there is none
   JunkStr = Right(SearchStr, \
            Len(SearchStr)-Instr(SearchStr + TargetCh, TargetCh))
   AfterChar = JunkStr
End Function

Const ShortStr = "The Wildcat & other bedside things"
Print ShortStr
Print Trim(AfterChar("&",ShortStr))
```

RIP Statement

Execute a RIP statement

Syntax

Rip *RipCommand* [*Parameters,...*]

Remarks

This statement will run any valid RIP command providing all of the correct parameters for the command are included. It is assumed that the RIP command is complete when passed for *wcCODE* to process as a RIP statement.

This command will only operate correctly if the command **RIPDetected** has returned true for the caller's system. For a list of the RIP commands supported see the appendix at the end of this manual.

Example

```
Rem Demo the Rip Circle statement

Dim Rad As Integer, Row As Byte, Col As Byte
Cls
Rad = 20
Row = 10 : Col = 40
If RipDetected Then
   Rip Circle Row, Col, Rad
End If
```

RIPDETECTED Function

Report if RIP graphics capability was detected at logon.

Syntax

RIPDetected {As True/False}

Remarks

The **RIPDetected** command is intended for use in a logon program and it will return **True** or **False** according to what was found.

See Also

RIPEnabled

Example

```
Rem Check caller's RIP status

If RIPDetected Then
   Print "RIP detected!"
End If
```

RIPENABLED Function

Report whether the caller has RIP graphics enabled.

Syntax

RIPEnabled {As True/False}

Remarks

The **RIPEnabled** function will return **True** if RIP graphics have been selected otherwise it returns **False**.

See Also

RIPDetected

Example

```
Rem If you can do it then use RIP

If RIPEnabled Then
   Print "RIP enabled!"
End If
```

RMDIR Statement

Remove a directory from a disk.

Syntax

RmDir *Directory*

Remarks

The RmDir statement will remove a directory from the disk only if there are no files left in the specified directory. If an attempt is made to remove a directory which contains files or other sub-directories then the Err_Path error will be returned which can in turn be handled with the Catch statement.

See also

Del, FindFirst, FindNext, Kill, MkDir

Example

```
Rem Remove a directory complete with files

Function RemoveDir(tStr As String) As Integer
   Dim SRec As SearchRec, Result As Integer
   RemoveDir = False
   If Not InputYesNo("Sure you want " + tStr + " removed? ") Then
      Exit Function
   Else
      If tStr = "" Then
         Print "Cannot work on current directory!"
         WaitEnter
         Exit Function
      End If
      If Right(tStr,1) <> "\" Then
```

```
        tStr = tStr + "\"
    End If
    Result = FindFirst(tStr + "*.*", 0, SRec)
    While Result = 0
        Print Pad(tStr + SRec.Name, 16); "Gone."
        Del(tStr + SRec.Name)
        Result = FindNext(SRec)
    Wend
    tStr = Left(tStr, Len(tStr) - 1) : Rem Drop the \
    RmDir(tStr)
    RemoveDir = True : Rem Place here and it 'should' be true
  End If
  Catch Err_Path
  Print "Error removing directory!"
End Function
Dim DirName As String
Print "Directory to remove: ";
Input DirName
If RemoveDir(DirName) Then
  Print "Directory and files are gone!"
End If
```

RND Function

Return a pseudo-random number.

Syntax

Rnd[(*Num*)] {As Real}

Remarks

This type of random function is generally referred to as a pseudo random number generator. The number returned is dependent on the value originally passed to the function that can therefore never be regarded as a true random number.

The value returned by **Rnd** will be equal to, or greater than zero and less than one.

If the *Num* is not supplied or is greater than zero, then the next random number in a sequence is generated.

If the *Num* value is less than zero then the **Rnd** function returns the same random number every time for the same value

If *Num* is zero, the last random number generated is returned.

Example

```
Rem Demonstrate the various uses of Rnd

Print "Rnd(10) = "; Rnd(10)
Print "Rnd(0)  = "; Rnd(0)
Print "Rnd(-1) = "; Rnd(-1)
Print "Rnd(0)  = "; Rnd(0)
```

RTRIM Function

Trim trailing spaces from a string.

Syntax

RTrim(*String*) {As String}

Remarks

The value *String* is any valid string type expression which will have all of the trailing spaces removed from it. The term 'trailing spaces' refers to any spaces to the right of the last non-space character. The *String* passed to this function is not modified during the process.

See also

LTrim, Trim

Example

```
Rem Trim all of the blanks, right of the last character

Dim tStr(6) As String, Ctr As Integer
tStr(1) = "    A little ditty     "
tStr(2) = "  hello Dolly     "
tStr(3) = "I said hello Dolly"
tStr(4) = "It's been good to see       "
tStr(5) = "     you way back when      "
tStr(6) = "        err I don't know any more words"
For Ctr = 1 To 6 Step 2
  Print Pad(LTrim(RTrim(tStr(Ctr))),40);
  Print Trim(tStr(Ctr + 1))
Next Ctr
```

RUN Statement

Execute another *wcCODE* program.

Syntax

Run *FileName*

Remarks

This command will execute or run another *wcCODE* program named *FileName* and then drop back to the original one on termination. This allows you to effectively "shell" to another *wcCODE* program. If the program *FileName* is not found, then an Err_Run is reported which can be handled with the **Catch** statement. The file extension of .WCX is forced at run-time, so the program to be run, must have a .WCX extension for the process to work.

Due to memory limitations, it is strongly suggested that you do not nest too many **Run** statements. As each one shells to the next, the amount of available memory is reduced. If you run out of memory, the offending program is terminated and the program flow returns to the previous program. There is an Err_Run reported which can be handled with the **Catch** statement.

If possible, it is far better to **Chain** programs than to **Run** them, as the **Chain** process releases all available memory for the next program to use.

See also

Chain, $Include

Example

```
Rem Shell to another program

Dim KeyIn As String*1, Row As Byte, Col As Byte
Do
   Print "[L]..List of BBS  [O]..Orders  [Q]..Quit to main menu ";
   Row = CsrLin : Col = Pos
```

```
    Do
        KeyIn = UCase(Inkey)
    Loop Until KeyIn > ""
    Locate Row, Col : Print KeyIn;
    Select Case KeyIn
        Case "L"
            Run "WCLIST.WCX"
        Case "O"
            Run "WCORDERS.WCX"
        Case "Q"
            End
        Case Else
            Beep
    End Select
Loop : Rem No condition needed as "Q" exits program
```

SCANMESSAGES Statement

Scan messages in the message database.

Syntax

ScanMessages

Remarks

This statement functions exactly the same as if the user had selected [S] Scan message from the Message menu.

Example

```
Rem If the caller desires, then scan the messages

If InputYesNo("Do you want to scan the messages? (YN) ") Then
   ScanMessages
End If
```

SEARCHFILES Statement

Search the contents of the *Wildcat!* files database.

Syntax

SearchFiles

Remarks

The **SearchFiles** command provides the exact same response to the user as if they had selected the [S]..Search files menu option.

Example

```
Rem Activate the file search system with SearchFiles

SearchFiles
```

SEEK Statement

Seek to a position in a file.

Syntax

Seek [#]*FileNum*, *FilePos*

Remarks

The **Seek** function allows you to position the file pointer at any given record in a **Random** file or at any byte in a **Binary** file. It is also possible to **Seek** to a prescribed byte position within a text file.

See also

Get, Input, Print, Put

Example

```
Rem Modify the data created in the Put example

Type TempRecord
   Name As String*25
   Password As String*14
   Phone As String*15
   SecLevel As String*10
End Type
Dim tRec As TempRecord, SeekPos As Long
Dim Prompt As String
Cls
Open "MyUsers.DAT" For Random As #1 Len = Len(TempRecord)
Prompt = Str(LOF(1)) + " records. Get #: "
SeekPos = InputNumber(Prompt, 0, LOF(1))
If SeekPos > 0 Then
   Seek #1, SeekPos
```

```
    Get #1,,tRec
    Print "Name: "; tRec.Name
    Print "Password needs changing! "; tRec.Password; " ";
    Input tRec.Password
    tRec.Password = UCase(tRec.Password)
    Put #1, SeekPos, tRec : Rem The alternate use
End If
Close
```

SELECT CASE Statement

Selective branching of program flow.

Syntax

Select Case *Exp*

Case *Condition*[, *Condition* ...]

Statements...

[Case *Condition*[, *Condition* ...]

Statements...]

[Case Else

Statements...]

End Select

Remarks

The **Select Case** statement allows many different branching choices to be made from the one set of criteria for *Exp*.

The value *Exp* can be any valid internal data type of **String**, **Byte**, **Integer**, **Long** or **Real**.

The *Condition* can be any valid **True** or **False** result to cause branching to the *Statement* statements. The *Condition* need *not* be an absolute **True** / **False** result but it may also be a range that the criteria *Exp* falls into.

The **Case Else** construct is an optional catch-all for everything that does not meet any of the preceding **Case** *Condition* decisions. This construct is optional but you will find you use it in almost all of your **Select Case** procedures.

See also

Else, If...Then

Example

```
Rem make a selection and branch using Select Case

Dim SelCh As String*1, SelInt As integer
Dim DosStr(2) As String, SN As String
SN = ProgName : DosStr(1) = "List.COM"
DosStr(2) = "Type " + Left(SN,Instr(SN,".") - 1) + ".WCC"
Do
   Print "[V]........View file          [1]......Dos Shell 1"
   Print "[Q].....Quit to menu          [2]......Dos shell 2"
   Print "Selection [V Q 1 2] : "; : Input SelCh
   Select Case SelCh
      Case Is = " " : Rem Boolean decision
         Print "Enter more than a blank space please!"
         Beep
      Case "V", "v"
         ViewFile
      Case "Q", "q"
         Exit Do
      Case "1" To "2" : Rem Range for decision
         Shell DosStr(Val(SelCh))
      Case Else : Rem If all else fails
         Print "Invalid selection."
         Beep
   End Select
Loop
```

SEND Statement

Output data to the communication port.

Syntax

Send *String*

Remarks

The **Send** command will send the *String* of characters to the COM port.

Note!

Unlike the **Print #***FileNum* command this does not append the carriage return and line feed characters to the string. If your program needs a CR/LF to follow the *String* then you must append it.

Example

```
Rem Reset a Hayes compatible MODEM

Delay(3)
Send "+++" + Chr(13)
Delay(3)
Send "ATZ" + Chr(13)
```

SENDFILE Function

Send a file to the caller, just as with download.

Syntax

SendFile(*Filename*) {As True/False}

Remarks

The **SendFile** function will send the *FileName* file to the caller currently on line. It will return a **True** or **False** result when finished.

See also

Download, Upload

Example

```
Rem Send a file after confirming the caller is ready

Dim KeyIn As String*1
Const EnterKey = Chr(13)
Send "Press [Enter] when ready for the file"
Do
   KeyIn = InComm
Loop Until KeyIn > ""
If KeyIn = EnterKey Then
   PushCommand("Y") : Rem Answers the SendFile question
   If SendFile("ORDERS.INF") Then
      Print "File sent OK"
   Else
      Print "SendFile failed!"
      WaitEnter
   End If
End If
```

SENDPAGE Statement

Page a node and include a message.

Syntax

SendPage *NodeNum*, *String*

Remarks

The **SendPage** command allows your program to take a *String* of data and pass it off to another *NodeNum* node.

The caller will not receive the **SendPage** text during uploads or downloads until the process has terminated.

See also

PageSysop, PageUser

Example

```
Rem Get a node number then page the caller

Dim PageStr As String, NodeNum As Integer
WhoIsOnLine : Rem We need their node number first
Print "Message: ";
Input PageStr
NodeNum = InputNumber("To which node? ", 1, MaxNode)
SendPage NodeNum, PageStr
```

SGN Function

Return a value indicating the sign of a number.

Syntax

Sgn(*Expr*) {As Real}

Remarks

The function will return a value of 1 if the expression *Exp is* positive and greater than zero, a value of -1 if the number is negative and less than zero, and zero if the *Exp* is itself equal to zero.

See also

Abs

Example

```
Rem Return the sign of various values

Print "Sgn -5 = "; Sgn(-5)
Print "Sgn 0  = "; Sgn(0)
Print "Sgn 3  = "; Sgn(3)
```

SHELL Statement

Run a DOS command or executable program.

Syntax

Shell [[*Command*], *Directory*]

Remarks

The *Command* can be a string expression with the desired executable program's name or command to process. Alternately the *Command* section can be a string literal enclosed between quotes.

The Command passed to the DOS Shell can be a direct DOS command, the name of an .EXE or .COM program to run or the name of a batch file with the .BAT extension.

If an executable file name is passed to DOS with the Shell command, the program would normally be run from the caller's node directory. By specifying the optional *Directory* this will force *Wildcat!* to change to that directory before running the program, and return after completion.

If the **Shell** command is issued without the optional *Command* then the system will **Shell** out to DOS and stay there until the DOS-Exit command is issued via the local keyboard to close the **Shell** and return to your program.

See also

End

Example

```
Rem Run a DOS program and a DOS command

Dim SelCh As String*1, SelInt As integer, DosStr(2) As String
Dim PN As String : PN = ProgName : DosStr(1) = "List.COM"
DosStr(2) = "Type " + Left(PN, Instr(PN, ".") - 1) + ".WCC"
Do
```

```
    Print "[V]..View file  [1]..Dos Shell 1  [R]..Read file"
    Print "[C]..Chg Setgs. [2]..Dos shell 2  [Q]..Quit to menu"
    Print "Selection [V R C Q 1 2] : "; : Input SelCh
    Select Case SelCh
        Case "V", "v"
            ViewFile
        Case "R", "r"
            ReadTextFile
        Case "C", "c"
            ChangeSettings
        Case "Q", "q"
            Exit Do
        Case "1" To "2"
            Shell DosStr(Val(SelCh))
        Case Else
            Print "Non-valid selection." : Beep
        End Select
Loop
```

SIN Function

Return the sine of an angular numeric expression.

Syntax

Sin(*AngleVal*) {As Real}

Remarks

The argument *AngleVal* is an angle expressed in radians. Radians can be converted to degrees by multiplying by 180/pi where pi is approximately 3.14159.

See also

Atn, Cos, Tan

Example

```
Rem Print the sine of a value

Print "The sine of 1 radian is equal to "; Sin(1)
```

SORTADD Statement

Add an element to the sorting system.

Syntax

SortAdd *String*, *SortIndex*

Remarks

The **SortAdd** command will add the *String* to the sort table. The *SortIndex* is the array subscript or index of the data being passed. The data should be in an array as the sort routine returns a numeric value associated with the data, that is the index into that array. Alternately the data could be in a data file with *SortIndex* as the record number.

This function will create a temporary file, if the size of the sort approaches the available memory. The file, if it is created, will be removed by **SortNext** or **SortPrev** returning a negative-one result, indicating the end of the sort has been reached.

The example program is in four parts and unlike most other examples in this manual, only the first will run without error on its own. To make use of the entire example, you will need to enter all four parts for a working program. This approach was adopted as all components of the sort process are inextricably tied.

See also

SortStart, SortNext, SortPrev

Example

```
Rem An example in four parts, part 2, add to the sort
Rem 1-SortStart, 2-SortAdd, 3-SortNext, 4-SortPrev
Function AddToSort As Integer
    Dim SortItems As Integer
    SortItems = 0
    For Ctr = 1 To MaxCtr
```

```
        If Trim(DataArr(Ctr)) > "" Then
            SortItems = SortItems + 1
            SortAdd(DataArr(Ctr), Ctr)
        End If
    Next Ctr
    AddToSort = SortItems
End Function
```

SORTNEXT Function

Return a positioning reference to the next item in sorted order.

Syntax

SortNext {As Long}

Remarks

The **SortNext** command will return the index number, or subscript of an array, of the next item in the sort table in a forward, or ascending-order or direction.

The **SortNext** function will dispose of itself when the last item in the sort system has been returned. The completion of the sort is indicated by **SortNext** returning a value of negative-one.

The example program is in four parts and unlike most other examples in this manual, only the first will run without error on its own. To make use of the entire example, you will need to enter all four parts for a working program. This approach was adopted as all components of the sort process are inextricably tied.

See also

SortAdd, SortPrev, SortStart

Example

```
Rem An example in four parts, part 3, retrieve forward
Rem 1-SortStart, 2-SortAdd, 3-SortNext, 4-SortPrev

Function GetTheNext As String
   Dim tSortRet As Long : tSortRet = SortNext
   If tSortRet > 0 Then
      GetTheNext = DataArr(tSortRet)
```

```
    Else
        GetTheNext = ""
    End If
End Function
```

SORTPREV Function

Return a reference to the previous item in sorted order.

Syntax

SortPrev {As Long}

Remarks

The SortPrev command will return the index number, or subscript of an array, of the next item in the sort table in a backward, or descending-order or direction.

The SortPrev function will dispose of itself when the last item in the sort system has been returned. The completion of the sort is indicated by SortNext returning a value of negative-one.

The example program is in four parts and unlike most other examples in this manual, only the first will run without error on its own. To make use of the entire example, you will need to enter all four parts for a working program. This approach was adopted as all components of the sort process are inextricably tied.

See also

SortAdd, SortNext, SortStart

Example

```
Rem An example in four parts, part 4, retrieve backward
Rem 1-SortStart, 2-SortAdd, 3-SortNext, 4-SortPrev

Function GetThePrev As String
   Dim tSortRet As Long : tSortRet = SortPrev
   If tSortRet > 0 Then
      GetThePrev = DataArr(tSortRet)
   Else GetThePrev = ""
   End If
End Function
```

SORTSTART Statement

Initialize the sorting system.

Syntax

SortStart

Remarks

The SortStart command will set up the sorting system ready to accept the data to be sorted.

The example program is in four parts and unlike most other examples in this manual, only the first will run without error on its own. To make use of the entire example, you will need to enter all four parts for a working program. This approach was adopted as all components of the sort process are inextricably tied.

See also

SortAdd, SortNext, SortPrev

Example

```
Rem An example in four parts, part 1, Set things up
Rem 1-SortStart, 2-SortAdd, 3-SortNext, 4-SortPrev

Const MaxCtr = 40
Dim DataArr(MaxCtr) As String*30, ForwardDir As Integer
Dim RetStr30 As String*30, Ctr As Long
Declare Function AddToSort As Integer : Rem Part 2
Declare Function GetTheNext As String : Rem Part 3
Declare Function GetThePrev As String : Rem Part 4
DataArr(1) = "Jim" : DataArr(2) = "Scott"
DataArr(3) = "Rick" : DataArr(4) = "Greg"
SortStart
```

```
ForwardDir = InputYesNo("Return in ascending order ")
If AddToSort > 0 Then
   Do
      If ForwardDir Then
         RetStr30 = GetTheNext
      Else
         RetStr30 = GetThePrev
      End If
      If RetStr30 > "" Then Print RetStr30
   Loop Until RetStr30 = ""
End If
```

SOUND Statement

Generate sound from the speaker.

Syntax

Sound *Freq, Dur*

Remarks

The **Sound** Statement will generate a sound from the speaker with the *Freq* Frequency in cycles per second (Hertz, Hz) and a duration of *Dur* in seconds.

This sound operates only at the local console and is not passed to the caller's system.

See also

Beep

Example

```
Rem Make a sound at the local console only

Sub AlarmBell
    Sound 3000, 1
    Delay 0.25
    Sound 600, 1
End Sub

Print "@F4@Warning"; DefColor
Do
  AlarmBell
Loop Until Inkey <> ""
```

SPACE Function

Return a string of blank spaces.

Syntax

Space(*Num*) {As String}

Remarks

This function will return a string filled with *Num* number of spaces and is exactly the same as the **Spc** function. Both are included to maintain uniformity with the BASIC language on which *wcCODE* is based.

See also

Spc, Tab

Example

```
Rem Neatly pad prompts and data display

Dim FirstN As String, LastN As String
Print LeftPad("First name: ", 18); : Input FirstN
Print LeftPad("Last name: ", 18); : Input LastN
Print Pad(FirstN,20); Space(5); UCase(LastN)
```

SPC Function

Returns a string consisting of a specified number of spaces.

Syntax

Spc(*Num*) {As String}

Remarks

This function will return a string filled with *Num* number of spaces and is exactly the same as the **Space** function. Both are included to maintain uniformity with the BASIC language on which *wcCODE* is based.

See also

Space, Tab

Example

```
Rem Neatly pad prompts and data display

Dim FirstN As String, LastN As String
Print LeftPad("First name: ", 18); : Input FirstN
Print LeftPad("Last name: ", 18); : Input LastN
Print Pad(FirstN,20); Spc(5); UCase(LastN)
```

SQR Function

Return the square root of a numeric expression.

Syntax

Sqr(*Expr*) {As Real}

Remarks

This function will return the square root of the *Exp* numeric value. The *Expr* value must be equal to or greater than zero otherwise an **Err_Math** error will be generated which can be handled with the **Catch** statement.

See also

Exp, Log

Example

```
Rem Test the Sqr function for rounding error

Dim tLong As Long

tLong = 16 * 16
Print "16 squared = "; tLong
Print "and the square root of "; tLong; " is "; Sqr(tLong)
```

STATIC Statement

Maintain the value of defined variables in a subroutine or function.

Syntax

Static *Name* **As** *Type* [, *Name* **As** *Type*...]

Static *Name*([*Lo* **To**] *Hi*) **As** *Type* [, *Name*([*Lo* **To**] *Hi*)] **As** *Type*...]

Remarks

The **Static** declaration is used to preserve the values of specific variables within a subroutine or function. Normally all variables defined within a **Sub** or **Function** are discarded on exiting the routine. In some cases it can be beneficial to keep the values of some internal variables for subsequent calls to use. These variables are declared using the **Static** statement instead of the familiar **Dim** statement.

As with **Dim**, all variables *must* be declared before they can be used. Each variable must have an associated data type assigned to it by using the **As** *Type* declaration which must be one of **Byte**, **Integer**, **Long**, **Real**, **String**.

If variables are declared on the same text line they must be separated by commas and different data types may be freely mixed on the one line. If separate lines are preferred then the **Static** statement must be used at the beginning of each text line.

The second construct is for declaring arrays. Arrays are accessed by their subscripts, the value in between the parentheses. When declaring arrays with the **Static** statement the '*Lo* **To**' construct can be omitted as all arrays will default to zero based if no *Lo* bound is declared.

The only time a '*Lo* **To**' statement would be useful is when dealing with a small range of array elements but with unusually high numbers. For instance if for some reason you wanted to reference a group of array elements starting with a *Lo* of 1200 and a *Hi* of 1230 it would waste a whole lot of memory by not declaring the lower bounds. The elements 0 through 1199 would never be used in such an example.

See also

Dim

Example

```
Rem Static variables are preserved while the program is active

Function CapName(tStr As String, TimesUsed As Integer) As String
   Dim CapStr As String
   Static NamesCapped As Integer
   CapStr = LCase(tStr) : Rem Copy, don't change the original
   CapName = UCase(Left(CapStr, 1)) + \
             Mid(CapStr, 2, Len(CapStr) - 1)
   NamesCapped = NamesCapped + 1
   TimesUsed = NamesCapped
End Function

Dim NameStr As String, Ctr As Integer
While Ctr < 3
   Print "Enter your name "; : Input NameStr
   Print "Hello "; CapName(NameStr,Ctr);
   Print "The Function CapName has now been used "; Ctr; " times"
Wend
```

STATISTICS Statement

Display the system statistics.

Syntax

Statistics

Remarks

The Statistics statement presents the user with the exact same response as the [S]..System statistics menu option.

Example

```
Rem A Wildcat! command to present the statistics information

Dim KeyIn As String
Print "Select [S]-System stats or [Q]-Quit back to menu (SQ) ";
Do
  KeyIn = UCase(Inkey)
Loop Until Instr("QS",KeyIn ) <> 0
If KeyIn = "S" Then Statistics
```

STOP Statement

Terminate execution of a program.

Syntax

Stop

Remarks

The **Stop** statement immediately terminates the currently executing program and returns to normal terminal operation where it displays a message. It may be executed at any time. Any currently open files are closed before the program ends.

The **Stop** statement may be used anywhere within a program, subroutine or function to immediately terminate it. It will display a message indicating the program was terminated prematurely.

See also

End

Example

```
Rem Stop reports to the Sysop and terminates the program

Dim Ctr As Integer, PwStr As String*5
For Ctr = 1 To 3
   Print "Enter your password "; Ctr; ": ";
   Input PwStr
   If UCase(PwStr) <> "SYSOP" Then
      Stop : Rem Terminate the process right now!
   End If
Next Ctr
Print "Ok so far so good"
```

STR Function

Convert a numeric value to a string.

Syntax

Str(*Num*) {As String}

Remarks

The **Str** command is the reverse of the **Val** function as it takes a numeric value. With either a numeric value or a variable it produces a string representation of the value.

It can sometimes be useful to extract a single digit or a series of digits from within a number and the **Str** function makes this an easy exercise.

See also

Hex, Oct, Val

Example

```
Rem Select a single digit from a number

Function PasswordVal(tStr As String) As Integer
   Dim Ctr As Byte, Accum As Long
   Accum = 0
   For Ctr = 1 To Len(tStr)
      Accum = Accum + (Asc(Mid(tStr,Ctr,1)) XOr Ctr) * Ctr * 17
   Next Ctr
   PasswordVal = Val(Mid(Str(Accum),3,4))
End Function

Dim PW1 As String*10, PW2 As String*10
Print "Enter your password : "; : Input PW1
   Rem Pass by value because of string length in Dim
```

```
Print "value = "; PasswordVal((PW1))
Print "Enter again to verify : "; : Input PW2
   Rem Pass by value because of string length in Dim
Print "value = "; PasswordVal((PW2))
```

STRING Function

Return a string filled with specified characters.

Syntax

String(*Num*, *Char*)	{As String}
String(*Num*, *CharVal*)	{As String}
String(*Num*, *String*)	{As String}

Remarks

Using the first construct, the **String** function will return a string filled with *Num* number of *Char* characters. The *Char* can be either a literal character in quotes or a variable containing the string-version of the character.

The second use of **String** is almost identical to the first with the exception that *CharVal* is a decimal value for the character to replicate.

The third structure of the **String** function takes the first character of the *StringVar*, string variable or literal and fills the return string with its value character.

See also

Chr, Space, Spc, String Type

Example

```
Rem String with a variety of formats

Const sLen = 65
Dim Lines As String*3
Lines = "-+="
Print String(sLen, Lines)
Lines = Right(Lines,2)
Print "Name"; String(8, 46); "  ";
```

```
Print "Address"; String(27, 46); "  ";
Print "Phone"; String(10, 46);
Print
Print String(sLen, Lines)
Print String(sLen, "=")
```

STRING Type

An internal data type for storing characters.

Syntax

Dim *Name* As String [**Length*]

Remarks

Variables of the **String** type hold the entire range of characters that can be entered via the keyboard. All ASCII values in the range of zero to 255, can be stored in strings after converting with the **Chr** function. The term string refers to a string of characters hence its type name.

The maximum default length of a string is 32,767 characters but this can be optionally shortened by using a *Length* identifier to restrict the maximum size to a pre-defined number of characters.

The string length designators cannot be used in the parameter list or header of a Sub or Function. Combining this and the strict type checking used in *wcCODE*, strings declared with fixed lengths can only be passed to a Sub or Function by value. This is done by enclosing the string to be passed in its own set of parentheses. For more information on passing by assignment or reference see the section **Call**.

Note!

The use of the length identifier is mandatory when declaring string variables within any user defined data type structures.

See also

Byte, Call, Dim, Integer, Long, Real, Type

Example

```
Rem The string data type formats

Type MyStringBag
    Food(50) As String*25
    Cola(5) As String*10
End Type
Dim TestString As String, MSB As MyStringBag
TestString = "Say something George!"
MSB.Food(1) = "Corn chips" : MSB.Cola(1) = "Pepsi"
Print TestString
Print MSB.Food(1); " & "; MSB.Cola(1)
```

SUB Statement

Define a subprogram that does not return a value.

Syntax

Sub *Name*[(*Arg* As Type[, *Arg* As Type]...)]

Statements...

End Sub

Remarks

A **Sub** is a subroutine or sub-program that performs a specific task. You would most often use a **Sub** if you found that you were typing the same or very similar code a second or more times in your source code. The preferred approach in such a case is to write a **Sub** that will perform the task simply by calling it, rather than duplicating code.

The *Name* is the name you assign to the subroutine which is usually descriptive of what the subroutine does. The argument(s) *Arg* is the name of a formal argument to the subroutine. These arguments are usually referred to as parameters. Each argument in the parameter list must have an associated type declaration using the **As** keyword and each must be separated by a comma.

If parameters are used in a **Sub** header then they must be enclosed in parentheses.

You may define local variables within your user-defined subroutine and you can also use **Static** declarations to preserve the value of individual variables across subroutine calls. Normally any variables defined in a subroutine or function are discarded on exiting but the use of the **Static** declaration will maintain the value of these specific variables while the program is running.

Note!

Subs *cannot* be defined within another subroutine definition. For a discussion on the use of a varying number of parameters, refer to the

section on **Declare** and also the preceding chapter under the sub-heading of **Subs** and **Functions**.

See also

Declare, Exit, Function, Stop, Static

Example

```
Rem A small wcCODE-usage file tracker

Sub TrackwcCODE
    Dim DT As DateTime, tStr As String
    CurrentDateTime DT
    tStr = "Program: " + ProgName + " "
    tStr = tStr + FormatDate(DT.D, "mm/dd/yy") + \
                  FormatTime(DT.T, " hh:mm")
    ActivityLog tStr
End Sub
TrackwcCODE
```

SYSOPACTIVITY Statement

Display the contents of the activity log file.

Syntax

SysopActivity

Remarks

The **SysopActivity** command will display the activity log for a specific node. The user is presented with the exactly the same response as the [A]..Activity log menu selection.

See also

SysopChat, SysopEraseLog, SysopEvents, SysopFiles, SysopNodes, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopActivity
End If
```

SYSOPCHAT Statement

Full screen Sysop chat mode.

Syntax

SysopChat

Remarks

The SysopChat statement produces exactly the same response as pressing the F6 key allowing the Sysop to chat with a caller. In the chat mode either a full screen or split screen option is available.

See also

SysopActivity, SysopEraseLog, SysopEventS, SysopFiles, SysopNodes, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopChat
End If
```

SYSOPDOWNLOAD Statement

A Sysop command to download a file.

Syntax

SysopDownload *FilePath*

Remarks

This statement has the same effect as the command that can be assigned to a menu function by using *MAKEMENU*. It allows the Sysop to perform a direct download of files that are not normally in the files database. It is intended for remote, online maintenance of the system.

The *FilePath* file path is the directory where the file is stored. On calling this statement the Sysop will be asked for the name of the file to transfer. Entering a blank file name will abort the process.

See also

SysopUpload

Example

```
Rem A direct Wildcat! command

SysopDownload "C:\WILDCAT\TEMFILE"
```

SYSOPERASELOG Statement

Erase the activity log for a particular node.

Syntax

SysopEraseLog

Remarks

The SysopEraseLog statement returns exactly the same response as the [E]..Erase activity log menu option.

See also

SysopActivity, SysopChat, SysopEvents, SysopFiles, SysopNodes, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopEraseLog
End If
```

SYSOPEVENTS Statement

Manage system events.

Syntax

SysopEvents

Remarks

The SysopEvents statement returns exactly the same response as the [M]..Manage events log menu option.

See also

SysopActivity, SysopChat, SysopEraseLog, SysopFiles, SysopNodes, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopEvents
End If
```

SYSOPFILES Statement

Access the Sysop's files menu.

Syntax

SysopFiles

Remarks

The **SysopFiles** statement returns exactly the same response as the [F]..File database menu option.

See also

SysopActivity, SysopChat, SysopEraseLog, SysopEvents, SysopNodes, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopFiles
End If
```

SYSOPNODES Statement

Node management.

Syntax

SysopNodes

Remarks

The SysopNodes statement returns exactly the same response as the [N]..Node management menu option.

See also

SysopActivity, SysopChat, SysopEraseLog, SysopEvents, SysopFiles, SysopStatus, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopNodes
End If
```

SYSOPSTATUS Statement

Display the status of the databases.

Syntax

SysopStatus

Remarks

The SysopStatus statement returns exactly the same response as the [S]..Status of database menu option.

See also

SysopActivity, SysopChat, SysopEraseLog, SysopEvents, SysopFiles, SysopNodes, SysopUsers

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopStatus
End If
```

SYSOPUPLOAD Statement

Sysop command to download a file.

Syntax

SysopUpload *FilePath*

Remarks

This statement has the same effect as the command that can be assigned to a menu function by using MakemEnu. It allows the Sysop to perform a direct upload of files that are not normally in the files database. It is intended for remote, online maintenance of the system.

The *FilePath* path is the directory where the file is to be stored. On calling this statement the Sysop will be asked for the name of the file to transfer. Entering a blank file name will abort the process.

See also

SysopDownload

Example

```
Rem A direct Wildcat! command

SysopUpload "C:\WILDCAT\NEWFILES"
```

SYSOPUSERS Statement

Display the user database.

Syntax

SysopUsers

Remarks

The SysopUsers statement returns exactly the same response as the [U]..User database menu option.

See also

SysopActivity, SysopChat, SysopEraseLog, SysopEvents, SysopFiles, SysopNodes, SysopStatus

Example

```
Rem A direct Wildcat! command

If User.SecLevel = "SYSOP" Then
   SysopUsers
End If
```

TAB Function

Move the cursor to a specified screen column.

Syntax

Tab(*Column*) {As String}

Remarks

This function when used in a **Print** statement will **Tab** (Tabulate) the cursor to the specified *Column* screen column. The screen column is absolute and not relative to the end of any previous **Print** statement if a trailing semicolon or comma had been used.

The **Tab** function does not actually move the cursor, but rather adds spaces to the end of the last printed item to position the cursor to the next **Tab** position.

See also

MoveLeft, MoveRight, Pad, Space, Spc, String

Example

```
Rem Columnar Tabbing

Print "Name"; Tab(20); "Address"; Tab(60); "Phone"
Print String(75,".")
Print
WaitEnter
```

TAN Function

Return the tangent of an angular numeric expression.

Syntax

Tan(*AngleVal*) {As Real}

Remarks

This function will return the tangent of the *AngleVal* numeric value. The *AngleVal* is the measurement of an angle expressed in radians. You can convert radians to degrees by multiplying by 180/pi (pi is approximately 3.14159).

See also

Atn, Cos, Sin

Example

```
Rem The Trigonometrical function

Print "The Tan of 33 is "; Tan(33)
```

TEMPPATH Function

Return the temporary path for the current node.

Syntax

TempPath {As String}

Remarks

The **TempPath** function will create a temporary work directory for the user node if it does not exist, then this function will return a path string complete with a trailing backslash.

See also

MkDir, RmDir

Example

```
Rem Make sure the TEMP directory exists by calling TempPath

Dim tDir As String
tDir = TempPath + "OLX-WORK"
MkDir(tDir)
```

TIMELEFT Function

Return the amount of time a user has left online.

Syntax

TimeLeft {As Integer}

Remarks

This function will report the amount of time a user has left on line. This value can be increased or decreased dynamically by using the **AddTime** function.

See also

AddTime, Timer, TimeLeft, TimeOnline

Example

```
Rem Warn the caller that their time is running short

Sub RunningShort
   If TimeLeft <= 5 Then
      Print "Time left online for "; User.Name; \
            " = "; TimeLeft
      Beep
   End if
End Sub
RunningShort
```

TIMEONLINE Function

Return amount of time user has been online.

Syntax

TimeOnline {As Integer}

Remarks

The **TimeOnline** statement returns the number of minutes the caller has been online for the current call.

See also

AddTime, Timer, TimeLeft

Example

```
Rem A direct Wildcat! command

Print "You have "; TimeLeft; " after being online for ";
Print TimeOnline
```

TIMER Function

Return the number of seconds since midnight.

Syntax

Timer {As Real}

Remarks

The Timer function will return the number of seconds since midnight as a real type value.

See also

AddTime, TimeLeft, TimeOnline

Example

```
Rem Use the timer function, wait and then proceed

Dim CutOut As Real, Col As Integer, KeyIn As String*1
Const Dur = 30
Print "You have "; Dur; " seconds to select [A] or [B] ";
CutOut = Timer + Dur
Col = Pos
Do
   Locate CsrLin, Col
   ClrEOL
   Print Int(CutOut - Timer);
   KeyIn = UCase(Inkey)
   If Instr("AB",KeyIn) Then Exit Do
Loop Until Timer > CutOut
```

TRIM Function

Trim leading and trailing blanks from a string.

Syntax

Trim(*String*) {As String}

Remarks

This command is a combination of **RTrim** and **LTrim** as it removes all leading and trailing blanks from the *String*.

See also

LTrim, RTrim

Example

```
Rem Make sure the string has no extra spaces

Dim TestStr As String
TestStr = "                    TESTING!                    "
Print "TestStr = "; Len(TestStr); " characters long now"
Print " but just "; Len(Trim(TestStr)); " characters now"
```

TYPE Declaration

Create a user defined data type.

Syntax

Type *UserTypeName*

Name **As** DataType

[[*Name* **As** DataType]...]

End Type

Remarks

The *UserTypeName* is the name you supply to describe your record structure which will consist of one or more variables or fields. The *Name* named variables can be any valid data type, including other user defined data types.

The only restriction on data types occurs with the **String** variable, as the length *must* be declared or a compiler error will result.

The declarations for your internal variables are exactly the same as for the **Dim** statement with the exception that the word **Dim** is not used.

To access the data in the newly created data type, the period is used to identify the individual variables.

Note!

There are a number of pre-defined data types that *Wildcat!* uses for storing relevant data on a user and the system. These data types can accessed in exactly the same way as this user defined data type. Refer to the Appendix in the back of this manual for more information.

See also

Dim

Example

```
Rem Define a user-type within a user type

Type MonthRecord
  Days As Byte
  Name As String*3
End Type
Type YearRecord
  Months(12) As MonthRecord
  ThisYear As Integer
End Type
Const MonthNames = "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"
Const MonthDays = "31,28,31,30,31,30,31,31,30,31,30,31"
Dim Ctr As Integer, Year As YearRecord
Year.Thisyear = 1994
For Ctr = 1 To 12
  Year.Months(Ctr).Name = Mid(MonthNames, ((Ctr - 1) * 4) + 1, 3)
  Year.Months(Ctr).Days = Val(Mid(MonthDays, \
                          ((Ctr - 1) * 3) + 1, 2))
Next Ctr
Print "For the year "; Year.ThisYear; " the days per month are..."
For Ctr = 1 To 12
  Print "The month of "; Year.Months(Ctr).Name;
  Print " has "; Year.Months(Ctr).Days; " days"
Next Ctr
```

UCASE Function

Convert all alphabetical characters in a string to upper case.

Syntax

UCase(*String*) {As String}

Remarks

This function will force all alphabetic characters in any string to upper case. It is the opposite of **LCase** which forces all alphabetic characters to lower case. The characters affected are 'A' through 'Z', 'a' through 'z' and any diacritical-marked characters in the international character-set in the current DOS configuration.

The *String* itself is not affected unless the function result is assigned to it, e.g. *String* = **UCase**(*String*).

See also

LCase

Example

```
Rem Capitalize a name

Dim NameStr As String
Print "Enter your first name please ";
Input NameStr
NameStr = LCase(NameStr)
NameStr = UCase(Left(NameStr,1)) + Mid(NameStr,2,Len(NameStr)-1)
Print "G'day "; NameStr
```

UPDATECONFS Statement

Update conferences.

Syntax

UpdateConfs

Remarks

The UpdateConfs statement returns exactly the same response as the [U]..Update conferences menu option.

Example

```
Rem A direct Wildcat! command

If InputYesNo("Before you go home shall "+ \
              "I update the conferences? ") Then
   UpdateConfs
End If
```

UPDATEFILE Function

Update a record in the file database.

Syntax

UpdateFile(*FileRec*) {As True/False}

Remarks

The **UpdateFile** function will update the file database with the *FileRec* file record information. The record must first be located with the GetFileInfo or GetNextFile function calls.

See also

GetFileInfo, GetNextFile

Example

```
Rem Indicate a new stored path

Dim fRec As FileRecord

Const FName = "ZEEUTIL1.ZIP", NewPath = "D:\LOOKHERE\"
If GetFileInfo(fRec, FName) Then

   fRec.StoredPath = NewPath

  UpdateFile(fRec)
End If
```

UPDATEMESSAGE Function

Update a message in the message database.

Syntax

UpdateMessage(*MessageHdr*, *Message*) {As True/False}

Remarks

The **UpdateMessage** function is used in conjunction with **GetMessage** and **GetNextMessage** to update the message database with a new or changed message. The *MessageHdr* is a data structure of the *Wildcat!* data type **MessageHeader**.

The *Message* part of this command will work in one of three ways. The first, it checks to see whether the contents of *Message* is a valid file name. If the file exists then that is regarded as containing the message text and its contents are added to the database.

If no file name is found matching the contents of *Message*, then the text *Message* contains is regarded as the message information to be added to the database.

The final option is that if Message is left blank, then the existing message-text is left untouched.

If the message is directed to a specific user by name and the **ToID** is zero, then *Wildcat!* will attempt to insert their Id from the database into the record. This will depend on two things, the user name must exist and there must be no duplicate user name in the database. If either of these conditions fail then a zero is placed in the **ToID** field of the record.

It is the responsibility of your program to ensure that sufficient information is contained in *Message* for it be regarded as a meaningful message in the database.

See also

AddMessage, **GetMessage**, **GetNextMessage**

Example

```
Rem Change the TO address of all messages

Dim mHdr As MessageHeader
Const OldTo = "ALL", NewTo = "SYSOP"
If GetMessage(mHdr,1) Then
   Do
      If mHdr.To = OldTo Then
         mHdr.To = NewTo
         UpdateMessage(mHdr,"")
      End If
   Loop Until Not(GetNextMessage(mHdr))
End If
```

UPDATEUSER Function

Update a record in the user database.

Syntax

UpdateUser(*UserRec*) {As True/False}

Remarks

The **UpdateUser** function is used in conjunction with the **GetUser** and/or **GetNextUser** to update the user database.

See also

Adduser, GetUser, GetNextUser

Example

```
Rem Change a user's SecLevel name

Dim uRec As UserRecord
Const OldSec = "FULLUSER", NewSec = "SYSOP"
uRec.Name = "Rick Heming"
If GetUser(uRec, uRec.Name) Then
    If Instr(OldSec, Trim(uRec.SecLevel)) > 0 Then
        uRec.SecLevel = NewSec
        UpdateUser(uRec)
    End If
End If
```

UPLOAD Statement

Accept an uploaded file from a caller.

Syntax

Upload

Remarks

The **Upload** statement returns exactly the same response as the [U]..Upload a file menu option.

Example

```
Rem Prepare to receive a file from the caller

If InputYesNo("Ready to send the file? ") Then
   Upload
End If
```

USERLIST Statement

Display the current user list.

Syntax

UserList

Remarks

The **UserList** statement returns exactly the same response as the [U]..User list menu option.

Example

```
Rem Display the current users

If InputYesNo("See the user list again? ") Then
   UserList
End If
```

VAL Function

Return the numeric value of a string.

Syntax

Val(*String*) {As Long}

Remarks

The **Val** function returns the value of a string of numeric characters. This function will begin evaluating the string at the first character and will consider the end reached on encountering the first non-numeric character in the string.

The plus and the minus characters are taken into consideration as valid numeric characters as is the dollar sign which is regarded as an identifier of Hexadecimal values.

The **Val** function does not support **Real** data types.

See also

Str

Example

```
Rem Various examples in the use of Val

Dim tVal As String*10
tVal = "1234"
Print Val(tVal)
Print Val(tVal) + 1234
Print Val("$B800 Good Hex!")
Print Val("B800 Not so good!")
```

VERSION Function

Return the current *wcCODE* version number as a string.

Syntax

Version {As String}

Remarks

This function simply returns the current version number for you to use in case you need to contact Technical Support or to keep track of your programs and under which version they will run.

See also

DosVersion

Example

```
Rem Save caller's request to a disk file

Dim Today As DateTime
CurrentDateTime(Today)
Open "TechSupp.TXT" For Append As #1
Print #1, "Date: "; FormatDate(Today.D, "mm/dd/yy")
Print #1, "Time: "; FormatTime(Today.T, "hh:mm")
Print #1, "Version: "; Version
Print #1, "Caller: "; User.UserId; "  ";
Print #1, User.Name
Print #1, "I need assistance with the user database"
Close #1
```

VIEWFILE Statement

View the contents of an archived file.

Syntax

ViewFile

Remarks

The **ViewFile** statement returns exactly the same response as the [V]..View a file menu option. This option allows the user to view the contents of compressed or archived files.

The caller will be presented with a prompt for the file name to view. If the name is left blank then **ViewFile** will terminate without error.

See also

ReadTextFile

Example

```
Rem Read text files or archived ones

Dim KeyIn As String*1
Do
   Print "Inspect file contents..."
   Print "[V]..View archived  [R]..Text files  "+ \
         "[Q]..Quit to menu (VRQ) ";
   Input KeyIn
   Select Case UCase(KeyIn)
      Case "V"
         ViewFile
      Case "R"
         ReadTextFile
   End Select
Loop Until (KeyIn = "Q") Or (KeyIn = "q")
```

WAITENTER Statement

Display a prompt and wait for the ENTER key.

Syntax

WaitEnter

Remarks

This command will display a prompt and wait for the user to press the [Enter] key.

See also

Delay, Pause

Example

```
Rem Display 10 lines of text then wait for the caller

Const PageLen = 10 : Dim tStr As String, Ctr As Byte
Cls : Ctr = 0
Open "Birthday.TXT" For Input As #1
While Not(EOF(1))
    Input #1, tStr
    Print tStr
    Ctr = Ctr + 1
    If Ctr = PageLen Then
        WaitEnter
        Ctr = 0
        Cls
    End If
Wend
Close
```

WEND Statement

The terminating counterpart of the While statement.

Syntax

Do While *Condition*

Statements...

Loop

While *Condition*

Statements...

Wend

Remarks

The **Wend** statement is the indicator that marks the end of the code that is to be repeated by the **While** statement. This command is only used with **While**, and only when **While** is used as a stand alone loop instruction and not part of the **Do...Loop** construct.

See also

Do...Loop, For...Next, While

Example

```
Rem Repeat a print statement

Dim Ctr As Integer
Do While Ctr < 10
   Print "This is meant for printing zero to nine: "; Ctr
   Ctr = Ctr + 1
Loop
While Ctr < 20
   Print "We are now running above the count of nine: "; Ctr
   Ctr = Ctr + 1
Wend
```

WHILE Statement

Perform a series of instructions until a condition has been met.

Syntax

Do While *Condition*

Statements...

Loop

While *Condition*

Statements...

Wend

Remarks

The **While** statement can be used either as a stand alone loop instruction or combined with the **Do...Loop** statement. In both cases it will cause a repeat of the enclosed *Statements...* until the *Condition* has been satisfied.

When used as a stand alone loop instruction the terminating statement **Wend**, is used to mark the end of the program block that is to be reapeated. When used in a **Do...Loop** construct the **Wend** is not used.

See also

Do...Loop, For...Next, Wend

Example

```
Rem Repeat a print statement

Dim Ctr As Integer
Do While Ctr < 10
   Print "This is meant for printing zero to nine "; Ctr
   Ctr = Ctr + 1
Loop
```

```
While Ctr < 20
    Print "We are now running above the count of nine: "; Ctr
    Ctr = Ctr + 1
Wend
```

WHOISONLINE Statement

Return the list of callers currently online.

Syntax

WhoIsOnline

Remarks

The **WhoIsOnline** statement returns exactly the same response as the [W]..Who is online menu option.

See also

UserList

Example

```
Rem Compare UL/DL ratios before allowing the command

Rem Add one to TotalDK to beat a divide by zero error
If ((User.TotalUK \ (User.TotalDK + 1)) * 100) >= 50 Then
   WhoIsOnline
Else
   Print "Command only available for 50% or better UL/DL ratios! :)"
End If
```

XOR Operator

Perform a logical exclusive Or operation between operands.

Syntax

Op1 XOr *Op2*

Remarks

The XOr operator will perform an exclusive Or operation on the two operands.

The *Op1* and *Op2* operands are logical expressions of any of the valid integer numeric data types, Byte, Integer or Long.

Note!

The use of bitwise operators requires a good understanding of Boolean logic. Space precludes us from going into greater detail in their use in this manual, but we include a small section in the Appendix along with truth tables for all of the operators.

See also

And, Eqv, Imp, Not, Or

Example

```
Rem Process a string with XOr to build a password value

Function PasswordVal(tStr As String) As Integer
   Dim Ctr As Byte, Accum As Long
   Accum = 0
   For Ctr = 1 To Len(tStr)
      Accum = Accum + (Asc(Mid(tStr,Ctr,1)) XOr Ctr) * Ctr * 17
   Next Ctr
   PasswordVal = Val(Mid(Str(Accum),3,4))
End Function
```

```
Dim PW1 As String, PW2 As String
Print "Enter your password : "; : Input PW1
Print "value = "; PasswordVal(PW1)
Print "Enter again to verify : "; : Input PW2
Print "value = "; PasswordVal(PW2)
```

YESCHAR Function

Return the local language character for the word Yes.

Syntax

YesChar {As String}

Remarks

This function will return the equivalent local language character to the letter 'Y' as the first character in the English word 'Yes'. This function is intended for evaluating Yes/No responses in your programs.

The **NoChar** and **YesChar** values are used by the **InputYesNo** function to provide multi-lingual support. They can also be used by your programs to determine the caller's response without having to know if they have elected to use foreign language prompts in *Wildcat!*.

See also

InputYesNo, NoChar

Example

```
Rem YesChar and NoChar can change according to the callers
Rem selected language but the code can handle it
Dim KeyIn As String*1
Print "Are you sure you want to do this? ";
Do
   KeyIn = UCase(Inkey)
Loop Until (KeyIn = YesChar) Or (KeyIn = NoChar)
```

Example programs included with *wcCODE*

These sample programs are provided in source code form. You are free to use and modify them to suit your preferences or run them exactly as-is. There are no royalties or fees attached to their use, but you cannot compile and distribute them in their current unmodified form.

You cannot make either the source code or the compiled executable programs available to anyone for download from your, or any other BBS.

If you modify any of these programs, then choose to distribute either the source code or the executable files, they must be sufficiently modified to be considered original works by the rulings of the United States Copyright Office (a department of the Library of Congress). This means that the program source code will have to have had **significant** modifications in order to comply with these rulings.

The sample programs and all of their associated files will be installed in the CODESAMP directory in the *Wildcat!* home directory.

Order entry system - WCORDER.WCC

This example Program is a reasonably generic order entry system for callers to use on your BBS. It is basically set out the way we at Mustang handle orders, but it could easily be modified to suit any style. The information required to run the program is contained in the control-file named ORDER.INF.

The source code is fully documented to make the programmers intentions better understood. Some time spent reading through the code could provide valuable insights into the power and flexibility of the *wcCODE* language for *Wildcat!*

The ORDER.INF file, is a three-line-structure, text file containing the names and prices of our products. This file also contains flag settings to indicate various conditions relevant to placing an order with Mustang. The format of this file could be easily changed, along with subsequent modifications to the ORDER.WCC program to conform to any specification you, as the

programmer desire. Naturally any changes to the .WCC file would require the program to be re-compiled.

This program makes good use of the formatted input features of *wcCODE* when accepting and verifying, to some degree, the validity of credit card numbers. It does not actually verify that the card is valid but rather the expected format for the card number is correct.

When an order is placed the caller can select from various methods of payment. The same applies to shipping via Mustang's preferred shipping agents for countries outside the United States. Once the order is complete and the caller is happy with it, it is added to a data file for later retrieval by the sales department for processing.

Files provided for WCORDER.WCX.

- WCORDER.WCC — Source code
- ORDER.INF — Text-form data file

BBS listing program - WCLIST.WCC

The WCLIST program is a comprehensive example of the *wcCODE* language and just what is possible to achieve with it. This program allows other Sysops to call into your system and maintain your list of BBS information for boards other than your own. Each caller who places an entry, or entries in the database, is required to maintain their relevancy within a ninety day period.

WCLIST.WCC

The complete program is spread across several include files and contains examples of a large number of the *wcCODE* functions and statements. In order to maintain a database with this potential, an efficient indexing method needed to be developed. The resultant indexing system which is based on a combination of key-hashing and the B-Tree structure has met this requirement well.

Another important part of the program is the sorting functions and WCLIST makes extensive use of this in-built facility. You would be well advised to

study the code contained in the accompanying files if you intend using the sort and indexing functions for examples of how to best use them.

The program has some interesting features built in to make life a little easier. When a caller is filling in the database form, on entering their phone number the area code is compared with the list in AREA.LST and the state is automatically added to the form. The same holds true for country-phone prefixes except that WCLIST fills in the name of the appropriate country.

PRMFILE.WCC

This file contains the necessary code to access a standard *Wildcat!* prompt file, so that multilingual support can be provided for all of the prompts used in the main program.

WCLISTP.WCC

This file contains the constant declarations for the program that relate to the items in the prompt file, WCLIST.PRM.

WCLIST1.WCC

This file contains the user-defined data type that is the basic structure of the main data file. It is supplied in its own file so that the record structure can be shared between all other parts of the source code without duplication. When and where it is required it is simply read in as an include file.

The WCLIST program should be made a menu option for your callers to add or update their information in the data files. The LOGON.WCX program should be used instead of the logon batch files and the last line in LOGON.WCC must chain to the WCLLOGON.WCX file. The WCLLOGON. program will check to ensure that it is less than 90 days since the caller updated their entries in your WCLIST.DAT database.

WCREPRT.WCC

This section of the program produces the reports and formats the output to a text file. This file could be compressed for online transmission to other BBSs or printed by redirecting file output to the printer using the DOS TYPE

or COPY commands. Alternatively the file could be added to the files database for a normal request and download.

Source code files supplied for WCLIST.

WCLIST.WCC	WCLIST1.WCC
WCLISTP.WCC	PRMFILE.WCC
WCLLOGON.WCC	WCLRPRT.WCC
LOGON.WCC	POSTCALL.WCC

Program files supplied for WCLIST

WCLIST.WCX	WCLLOGON.WCX
LOGON.WCX	POSTCALL.WCX

Data files created or used by WCLIST.WCX.

WCLIST.DAT	Main data file [created]
WCLIST.IX1	Primary index file [created]
WCLIST.IX2	Secondary index file [created]
AREA.LST	Phone area codes [supplied]
COUNTRY.LST	Phone country codes [supplied]
WCLIST.PRM	Prompts file [supplied]

Appendix

Humor is emotional chaos remembered in tranquility.

James Thurber

Wildcat!

Reserved Keywords for *wcCODE*

Keyword	Description
+ (Concatenation)	Concatenate two or more strings.
\ (Line-break)	Break a source code line.
ABS	Return the absolute value of a numeric value or expression.
ACTIVITYLOG	Add a string to the activity log.
ADDFILE	Add a file to the Files database.
ADDMESSAGE	Add a message to the Message database.
ADDTIME	Increase or decrease the caller's remaining time left on-line.
ADDUSER	Add a new caller's details to the User database.
AND	Perform a logical And operation between operands.
ANSIDETECTED	Report whether the caller's terminal has ANSI capabilities.
ANSIENABLED	Report whether the caller has ANSI enabled.
APPEND	Open a text or sequential file to append data.
ASC	Return the ASCII value of a character.
ATLEN	Return the length of a string with the @Codes stripped out.
ATN	Return the arctangent of an angular numeric expression.
BAUDRATE	Return the current connection baud rate.
BEEP	Sound the beeper.
BINARY	Open a file for binary access.
BULLETIN	Display a specific bulletin file.
BULLETINMENU	Show the bulletin menu.
BYTE	An internal data type for storing one byte values.
CALL	Execute a subroutine or function.
CALLERIDRESULT	Return the caller ID information.
CAPTURE	Open or close a capture file.

Keyword	Description
CARRIER	Report the state of the line carrier.
CARRIERCHECK	Turn automatic carrier detection on or off.
CASE	Used in the Select Case statement.
CATCH	Catch runtime errors.
CHAIN	Transfer control to another compiled *wcCODE* program.
CHANGESETTINGS	Change the caller's default settings.
CHECKBULLETINS	Check for new bulletins.
CHECKMAIL	Check for a caller's mail.
CHECKMAILATLOGON	Check for caller's mail at logon and wait for a response.
CHECKNEWSLETTER	Check for a new newsletter since the callers last logon.
CHR	Return the ASCII character for the supplied value.
CLOSE	Close a file or all files.
CLREOL	Clear from the cursor to the end of the current screen-line.
CLS	Clear the entire screen.
COLOR	Set the current terminal color.
COMMENT	Leave a comment to the Sysop.
CONST	Used to assign symbolic names to represent values.
COPYFILE	Copy a file to another directory or drive.
COS	Return the cosine of an angular numeric expression.
CSRLIN	Return the screen line number of the cursor position.
CURRENTDATE	Return the current date.
CURRENTDATETIME	Return the current date and time.
CURRENTTIME	Return the current time.
DECLARE	Forward declare a subroutine or function.
DEFCOLOR	Return the default color.

Keyword	Description
DEL	Delete a file from disk.
DELAY	Suspend program execution for a defined period of time.
DELETEMESSAGE	Delete a message
DIM	Declare variables of any data type.
DISPLAYFILE	Display the contents of a text file.
DISPLAYGROUPFILES	Display group files.
DISPLAYHELP	Display help file contents.
DISPLAYPROMPT	Display a prompt from the prompt file.
DISPLAYSTOPPED	Indicate if the caller selected Stop from the Pause command.
DISPLAYTEXTFILE	Display the contents of a text file.
DO	Repeatedly execute a block of statements.
DOOR	Run a Door program.
DOORMENU	Display the door menu.
DOSVERSION	Return the current DOS version.
DOWNLOAD	Download a file.
EDITMARKLIST	Edit the list of files marked for downloading.
ELSE & ELSEIF	Provide alternatives to an If-condition not being met.
ENCODEPASSWORD	Encode a password for the user database.
END	End a program.
ENTERMESSAGE	Enter a message to the message database.
ENVIRON	Return the contents of a DOS environment variable.
EOF	Determine if the end of a file has been reached.
EQV	Perform a logical Equivalence operation between operands.
ERROR	Force a defined runtime error.
EXISTS	Determine whether a file exists on disk.

Keyword	Description
EXIT	Exit immediately from a loop, subroutine, or procedure.
EXP	Return a value raised to a specified power.
FASTLOGON	Return a result on the caller's logon method.
FILEINFO	Display the extended description for a file.
FILESTATS	Display the statistics on file transfers.
FINDFIRST	Find the first file to match a file name specification.
FINDNEXT	Finds the next file in the set defined by the FindFirst function.
FIX	Return the integer portion of a Real value.
FOR	Execute a sequence of statements a specified number of times.
FORMATDATE	Format the Date data type.
FORMATTIME	Format the Time data type.
FREEFILE	Return the next available file number.
FUNCTION	Define a subprogram that will return a value.
GET	Read information from a file.
GETCOLOR	Return the current screen color.
GETCONFINFO	Get the details of a conference.
GETFILEAREA	Get the details of a file area.
GETFILEINFO	Get file record information.
GETMESSAGE	Get the message header details from the message database.
GETMESSAGETEXT	Copy the message text for a particular message to a text file.
GETNEXTFILE	Get the next file in sequence from the files database.
GETNEXTMESSAGE	Get next message in sequence from the message database.
GETNEXTUSER	Get the next user in sequence from the user database.
GETNODEINFO	Get a specific node information record.

Keyword	Description
GETSECPROFILE	Get a specific security profile.
GETUSER	Get a specified user's details.
GOODBYE	Gently log off a caller.
GOSUB	Branch unconditionally to the specified line number or label.
GOTO	Branch unconditionally to a specified line or label.
HANGUP	Hang up the modem without warning.
HELLOFILES	Display the Hello files to a caller.
HELPLEVEL	Set a caller's help level.
HEX	Convert a decimal value to its Hexadecimal equivalent.
IF	Make a decision on program flow.
IMP	Perform a logical Implication operation between operands.
$INCLUDE	Include another source code file.
INCOMM	Read a character from the communications port
INDEXLOOKUP	Initialize the index-retrieval system with a starting string.
INDEXNEXTKEY	Return the next key.
INDEXNEXTREF	Return the positioning information for the next exact-match key.
INDEXOPEN	Open an index file and initialize the index system.
INDEXQUERY	Apply Boolean logic to a multiple key search.
INDEXQUERYNEXT	Return the positioning information for the keys.
INKEY	Read a character from the keyboard, communications port, or file.
INPUT	Open a text or sequential file for the input of data.
INPUT	Read a line of text from the keyboard or a file.
INPUTMASK	Supply a prompt and read a formatted line of text.

Keyword	Description
INPUTNUMBER	Read numbers from the keyboard with range checking.
INPUTYESNO	Read a yes or no response from the keyboard
INSTR	Search for the occurrence of one string within another.
INT	Return the integer portion of a number.
INTEGER	An internal data type for storing small signed values.
JOIN	Join an existing conference.
KILL	Kill or delete a file.
LCASE	Convert all alphabetic characters in a string to lower case.
LEFT	Return a specified number of characters from a string.
LEFTPAD	Pad the left side of a string with blanks.
LEN	Return the number of characters in a string or other data type.
LET	Assign the value of an expression to a variable.
LISTFILES	List all available files in the files database.
LISTFILESDATE	List all available files according to date criteria.
LOC	Return the position of the open-file pointer.
LOCAL	Return true if the user is logged on locally.
LOCATE	Position the cursor on the screen.
LOCATEUSER	Display a users details.
LOF	Return the number of records in an open file.
LOG	Return the natural logarithm of a numeric expression.
LONG	An internal data type for storing large signed values.
LOOP	An integral part of the Do...Loop construct.
LTRIM	Trim leading spaces from a string.
MAXNODE	Return the maximum nodes for the installed version of *Wildcat!*.

Keyword	Description
MID	Retrieve a sub-string from within another string.
MKDIR	Make a new directory on the disk.
MOD	Return the remainder of a division between two values.
MOREPROMPT	Enable, disable or report the status of the Pause prompt.
MOVELEFT	Move the cursor left.
MOVERIGHT	Move the cursor right.
NAME	Rename or move a file.
NEWSLETTER	Display the Newsletter
NEXT	An integral part of the For...Next loop procedure.
NOCHAR	Return the local language character for the word No.
NODE	Return the current node number.
NOT	Perform a logical Not operation.
NUMBER	Storage area for the value of a Date or Time.
OCT	Convert a value to its octal representation.
OPEN	Open a file for data access.
OR	Perform a logical Or operation between operands.
OUTPUT	Open a text or sequential file for the output of data.
PAD	Pad a string with spaces on the right.
PAGESYSOP	Page the Sysop.
PAGEUSER	Page an on-line user.
PAUSE	Suspend execution until the caller responds.
POS	Return the column position of the cursor.
PRINT	Output data to the screen or a file.
PROGNAME	Return the name of the currently running program.
PROGPATH	Return path the of the currently running program.

Keyword	Description
PUSHCOMMAND	Load the keyboard buffer with keystrokes.
PUT	Write data to Random or Binary files.
QUESTIONNAIRE	Run a specified questionnaire.
QUESTIONNAIREMENU	Display the questionnaire menu.
QUOTEOFTHEDAY	Display a quote from the Quote-of-the-day file
RANDOM	Open a file for random access.
READMESSAGES	Read messages in the current conference.
READTEXTFILE	Read the contents of a text file.
REAL	An internal data type for storing signed floating point values.
REM	Include a remark or comment.
RESET	Close all open files.
RETURN	Return from a subroutine.
RIGHT	Return a specified number of characters from a string.
RIP	Execute a RIP statement
RIPDETECTED	Report if RIP graphics capability was detected at logon.
RIPENABLED	Report whether the caller has RIP graphics enabled.
RMDIR	Remove a directory from a disk.
RND	Return a pseudo-random number.
RTRIM	Trim trailing spaces from a string.
RUN	Execute another *wcCODE* program.
SCANMESSAGES	Scan messages in the message database.
SEARCHFILES	Search the contents of the *Wildcat!* files database.
SEEK	Seek to a position in a file.
SELECT CASE	Selective branching of program flow.
SEND	Output data to the communication port.

Keyword	Description
SENDFILE	Send a file to the caller, just as with download.
SENDPAGE	Page a node and include a message.
SGN	Return a value indicating the sign of a number.
SHELL	Run a DOS command or executable program.
SIN	Return the sine of an angular numeric expression.
SORTADD	Add an element to the sorting system.
SORTNEXT	Return a positioning reference to the next item in sorted order.
SORTPREV	Return a reference to the previous item in sorted order.
SORTSTART	Initialize the sorting system.
SOUND	Generate sound from the speaker.
SPACE	Return a string of blank spaces.
SPC	Returns a string consisting of a specified number of spaces.
SQR	Return the square root of a numeric expression.
STATIC	Maintain the value of defined variables in a subroutine or function.
STATISTICS	Display the system statistics.
STOP	Terminate execution of a program.
STR	Convert a numeric value to a string.
STRING	Return a string filled with specified characters.
STRING	An internal data type for storing characters.
SUB	Define a subprogram that does not return a value.
SYSOPACTIVITY	Display the contents of the activity log file.
SYSOPCHAT	Full screen Sysop chat mode.
SYSOPDOWNLOAD	A Sysop command to download a file.
SYSOPERASELOG	Erase the activity log for a particular node.

Keyword	Description
SYSOPEVENTS	Manage system events.
SYSOPFILES	Access the Sysop's files menu.
SYSOPNODES	Node management.
SYSOPSTATUS	Display the status of the databases.
SYSOPUPLOAD	Sysop command to download a file.
SYSOPUSERS	Display the user database.
TAB	Move the cursor to a specified screen column.
TAN	Return the tangent of an angular numeric expression.
TEMPPATH	Return the temporary path for the current node.
TIMELEFT	Return the amount of time a user has left online.
TIMEONLINE	Return amount of time user has been online.
TIMER	Return the number of seconds since midnight.
TRIM	Trim leading and trailing blanks from a string.
TYPE	Create a user defined data type.
UCASE	Convert all alphabetical characters in a string to upper case.
UPDATECONFS	Update conferences.
UPDATEFILE	Update a record in the file database.
UPDATEMESSAGE	Update a message in the message database.
UPDATEUSER	Update a record in the user database.
UPLOAD	Accept an uploaded file from a caller.
USERLIST	Display the current user list.
VAL	Return the numeric value of a string.
VERSION	Return the current *wcCODE* version number as a string.
VIEWFILE	View the contents of an archived file.
WAITENTER	Display a prompt and wait for the ENTER key.

Keyword	Description
WEND	The terminating counterpart of the While statement.
WHILE	Perform a series of instructions until a condition has been met.
WHOISONLINE	Return the list of callers currently online.
XOR	Perform a logical Exclusive Or operation between operands.
YESCHAR	Return the local language character for the word Yes.

@codes

@ codes are filled in with system information by *Wildcat!* when the file or prompt is displayed to the caller.

System Information

System Information is information pertaining to the current operational state of the BBS.

Macro	Notes
@BBS@	Name of BBS
@CONF@	Current conference name
@CONFOP@	Conference Sysop
@DATE@	Current system date
@ID@	Packet ID for mail door
@INCHAT@	Number of user active in *wcCHAT*
@LASTCALL@	Time of last caller
@NODE###@	Node ID status
@NODE@	Node ID number for current connection
@PREVON@	Name of last caller
@SUB##@	Substitution variables
@SYSOP@	Sysop's full name
@TCALLS@	Total calls received
@TFILE@	Total number of files
@TIME@	Current system time
@TMSG@	Total number of messages
@TUSER@	Total number of users
@VER@	*Wildcat!* version number
@XX@	Default color in Prompt file

User Record Information

User record information is extracted from the current caller's user file. You can use these macros to display information to the caller such as name, address, current file statistics, and so on.

Macro	Notes
@ACCBAL@	Account balance
@ADDR1@	Address 1
@ADDR2@	Address 2
@ALIAS@	Alias name
@BDATE@	Birthday
@CALLS@	Total number of logons
@CITY@	City
@COMPANY@	Company
@CPU@	Computer type
@DATA@	Data phone number
@DAYBYTES@	Daily download K
@DLKLIMIT@	Max daily download K limit
@DOWNK@	Total download K
@DOWNS@	Total number of downloads
@EXDATE@	Expiration date
@EXPDAYS@	Days until user's expiration date
@FIRST@	name of caller
@FROM@	Location of caller
@KBLEFT@	Download K left today
@LAST@	Last name of caller
@LASTNEW@	Last new files search date

Macro	Notes
@LASTON@	Date of last call
@LEFT@	Time remaining this call
@MAXDK@	Maximum download K per day
@MAXDL@	Downloads per day
@MAXRATIO@	Maximum download ratio
@MEMO@	Memo date
@MEMO1@	Comment 1
@MEMO2@	Comment 2
@MSGSLEFT@	Total number of messages posted
@NETBAL@	Netmail balance
@PHONE@	Phone number
@PROTOCOL@	Default transfer protocol
@SEC@	Security profile
@SINCE@	Date of first call
@STATE@	State
@TIMEON@	Time on system this call
@UPK@	Total upload kilobytes
@UPS@	Total number of uploads
@USER@	Full name
@USERID@	Unique UserID for caller
@ZIP@	Zip code

Connection variables

Connection variables are passed to *Wildcat!* at the time the current call was answered.

Macro	Notes
`@BAUD@`	Current baud rate
`@CALLID@`	Caller ID string from modem
`@CONNECT@`	Connect string from modem
`@MODEM@`	Name of modem from MDM file

Screen control codes

Screen control codes affect the way display files are shown to the caller. You can add or suppress screen pauses (the -Continue- prompt), allow or suppress the "Previous page" option when displaying files, and display portions of a file only to callers with a specific security profile.

Macro	Notes
`@BELL@`	Beeps the speaker
`@CLS@`	Clears the screen
`@ELSE@`	Else condition for IFSEC
`@ENDIF@`	Ends IFSEC
`@ENTER@`	Requests ENTER to Continue prompt
`@IFSEC=[sec]@`	Display text if security profile matches
`@LOGOFF@`	Log the user off
`@PAUSE@`	Pauses screen display
`@NOCODE@`	Turns off all further @ prompts in file
`@NOPAUSE@`	Turns off pause prompts
`@NOPREV@`	Turns previous mode off
`@NOSTOP@`	Turns off user's ability to abort file
`@STOP@`	Turns on user's ability to abort file

Color Codes

The color codes system follows the format @BF@ where B is the background color and F is the foreground color. Blinking is controlled via background choice.

Hex value	Screen Color	Background	Foreground
0	Black	Normal	Normal
1	Blue	Normal	Normal
2	Green	Normal	Normal
3	Cyan	Normal	Normal
4	Red	Normal	Normal
5	Magenta	Normal	Normal
6	Brown	Normal	Normal
7	White	Normal	Normal
8	Black	Blinks FG	Hi Intensity
9	Blue	Blinks FG	Hi Intensity
A	Green	Blinks FG	Hi Intensity
B	Cyan	Blinks FG	Hi Intensity
C	Red	Blinks FG	Hi Intensity
D	Magenta	Blinks FG	Hi Intensity
E	Brown	Blinks FG	Hi Intensity
F	White	Blinks FG	Hi Intensity

ANSI codes

ANSI codes are used for color and cursor placement in .SCR files. They should not ordinarily be used in .BBS or .RIP display files. A good screen drawing utility for creating .SCR files is THEDRAW, available for download from MSI HQ BBS.

Command	ANSI CODE	DOS COLOR #
ALL ATTR. OFF	ESC [0m	
BACKGROUND		
BLACK BG	ESC [40m	
BLACK FG	ESC [30m	0
BLINK ON	ESC [5m	add 128 to color #
BLUE BG	ESC [44m	
BLUE FG	ESC [34m	1 (bold = 9)
BOLD ON	ESC [1m	add 8 to color #
CANCEL ON (invisible)	ESC [8m	
CYAN BG	ESC [46m	
CYAN FG	ESC [36m	3 (bold = 11)
Erase display	ESC [2J	
FOREGROUND		
GREEN BG	ESC [42m	
GREEN FG	ESC [32m	2 (bold = 10)
MAGENTA BG	ESC [45m	
MAGENTA FG	ESC [35m	5 (bold = 13)
RED BG	ESC [41m	
RED FG	ESC [31m	4 (bold = 12)

Command	ANSI CODE	DOS COLOR #
REVERSE VIDEO ON	ESC [7m	
UNDERSCORE ON	ESC [4m	TTL Mono only
WHITE BG	ESC [47m	
WHITE FG	ESC [37m	7 (bold = 15)
YELLOW BG	ESC [43m	
YELLOW FG	ESC [33m	6 (bold = 14)

Compile-time errors

Err	Message	Description
1	Error opening input file	The compiler cannot open an input file, either the main one or an $INCLUDE file.
2	Error creating output file	The compiler cannot create the program output file for some reason. The disk might be full or the .WCX could be locked by another user if connected to a network.
3	Syntax error	This indicates a generic problem with the wording of your program. Check in the reference section for the statement you are trying to use. Make sure the spelling and the structure are correct.
4	End of statement expected	You most likely have information following a command that should not be part of the statement on that line.
5	Number too large	Integer numbers must be in the range -2147483648 to 2147483647.
6	Error in floating point number	The compiler has encountered a number it has determined should be a floating point number, but it is not in a correct format.
7	String not terminated	A double quote character appears on the line without a closing double quote character.
8	= expected	An equals sign is expected here. You may get this error if you misspell a program statement. The compiler will not recognize the word and assume that it is a variable you are trying to assign a value to.
9	(expected	An open parenthesis is expected here. You may be trying to call a function with no arguments when it actually requires arguments.
10	) expected	A close parenthesis is expected here.
11	, expected	A comma is expected here. You may be trying to call a function with fewer parameters than it requires.

Err	Message	Description
12	Type mismatch	This indicates you are using a variable or expression that is the wrong data type. You may be trying to assign a string value to a numeric data type or vise versa. You could also be attempting to pass a fixed-length string to a Sub or Function as a parameter. Instead pass the fixed-length string by value, by enclosing it in its own set of parentheses.
13	THEN expected	The THEN keyword is expected after the condition in an IF statement.
14	ELSE not allowed here	The ELSE keyword is only permitted after an IF or ELSEIF statement. If an ELSE for the current IF has already been encountered, another ELSE clause is not permitted.
15	ELSEIF not allowed here	The ELSEIF keyword is only permitted after an IF statement. It is not permitted after an ELSE clause.
16	UNTIL or WHILE expected	The UNTIL or WHILE keyword is expected after a DO or LOOP keyword.
17	LOOP not allowed here	The LOOP keyword is only permitted after a matching DO statement.
18	EXIT DO not within DO ... LOOP	The EXIT DO statement must occur within a DO ... LOOP construct.
19	EXIT FOR not within FOR ... NEXT	The EXIT FOR statement must occur within a FOR ... NEXT loop.
20	DO, FOR, SUB, or FUNCTION expected	The EXIT keyword must be followed by DO, FOR, SUB or FUNCTION.
21	Duplicate label declaration	The label you are trying to define has already been defined within the current program block.
22	LOOP expected	The LOOP keyword is expected at the end of a DO ... LOOP.
23	END IF expected	The END IF keywords must follow an IF statement.

Err	Message	Description
24	WEND expected	Each WHILE statement must have a matching WEND statement.
25	Variable expected	A variable is expected wherever an assignment occurs.
26	WEND not allowed here	Each WEND statement must have a matching WHILE statement.
27	Function expected	This error means that you tried to call a subroutine as a function. Since subroutines cannot return a value, this is not possible.
28	Subroutine name expected	A subroutine name is expected after the CALL keyword.
29	FUNCTION or SUB expected	Either the FUNCTION or SUB keyword is expected after the DECLARE keyword.
30	Identifier expected	An identifier is expected here. This generally means that you are trying to declare an identifier but none was found.
31	Type character illegal for SUB	Subroutine declarations, since they do not return a value, cannot have a type specification.
32	AS expected	The AS keyword is expected here. Since you need an AS keyword, the preceding identifier needs a type declaration.
33	Type name expected	A data-type name is expected after the AS keyword.
34	Type specification illegal	A type specification is not allowed in the indicated case.
35	Parameter list type mismatch	A function declaration and the corresponding definition must match in the number, type and order of arguments.
36	Parameter list has fewer items than declaration	A function declaration and the corresponding definition must match in the number, type and order of arguments.

Err	Message	Description
37	Parameter list has more items than declaration	A function declaration and the corresponding definition must match in the number, type and order of arguments.
38	Nested FUNCTIONs or SUBs not allowed	User defined subroutines and functions must not be declared without correctly ending the previous subroutine or function.
39	EXIT FUNCTION/SUB not allowed in main code	EXIT FUNCTION and EXIT SUB can only appear within a subroutine or function, not the main program body.
40	AS not allowed here	This error is caused by using the AS keyword in a place where it is not permitted.
41	Duplicate definition	The subroutine or function you are trying to define has already been defined once before.
42	Duplicate identifier	The identifier you are trying to declare has already been used as a variable, constant, subroutine, function name or type name.
43	Constant expression required	A constant expression is required here. For example, when declaring an array, the array dimension must be a constant.
44	Array variable expected	This error is most often caused by trying to assign a value to a subscripted variable that has not been declared as an array.
45	Integer expression required	An integer expression is required here. For example, when dimensioning an array, an integer value is required for the dimension value.
46	Invalid array bounds	The lower bound of an array must be less than or equal to the upper bound.
47	Reserved	
48	File mode expected	One of the keywords INPUT, OUTPUT, APPEND, BINARY, or RANDOM must follow the FOR keyword in an OPEN statement.

Err	Message	Description
49	Reserved	Reserved for future use.
50	Reserved	Reserved for future use.
51	LEN expected	The LEN keyword is expected at the end of an OPEN statement.
52	TO expected	The TO keyword is expected after the starting value in a FOR statement.
53	NEXT expected	The NEXT keyword is expected after a FOR loop.
54	NEXT not allowed here	The NEXT keyword must be preceded by a matching FOR statement.
55	NEXT without FOR	You have used a NEXT keyword with a variable that does not match the variable used in the corresponding FOR keyword.
56	CASE expected	The keyword CASE is required after the keyword SELECT.
57	END SELECT expected	The END SELECT keywords must follow a SELECT CASE block.
58	CASE not allowed after CASE ELSE	The CASE ELSE clause in a SELECT CASE statement must be the last case clause in the statement.
59	ELSE or IS expected	If a CASE keyword in a SELECT CASE statement is not followed by an expression, it must be followed by an ELSE or IS keyword.
60	Statements not allowed here	Statements are not allowed between a SELECT CASE statement and the first CASE clause. Also, statements are not permitted within a user defined type declaration.
61	=, <>, <, <=, >, or >= expected	In a CASE clause within a SELECT CASE statement, the keyword IS must be followed by one of the relational operators listed.

Err	Message	Description
62	END SUB or END FUNCTION expected	The END SUB or END FUNCTION keywords are expected here.
63	EXIT FUNCTION expected	If the EXIT keyword appears within a function, it should be an EXIT FUNCTION statement.
64	EXIT SUB expected	If the EXIT keyword appears within a subroutine, it should be an EXIT SUB statement.
65	END FUNCTION expected	The END FUNCTION keywords must follow a function declaration.
66	END SUB expected	The END SUB keywords must follow a subroutine declaration.
67	Undeclared identifier	Identifiers must be declared before using them. This error is most often caused by using a variable before having a DIM statement that declares it.
68	END TYPE expected	The END TYPE keywords must follow a user defined type declaration.
69	Duplicate type definition	The type name you are trying to declare has already been declared.
70	Type character not allowed	A type specification character is not allowed here.
71	User defined type variable expected	You are trying to use the dot notation to access a field of a user defined type, but the variable to the left of the dot is not a user defined type variable.
72	Field name expected	A field name must appear after a period when accessing a particular field of a user defined type.
73	Dynamic strings not allowed in user defined types	Variable sized strings are not permitted in a user defined type declaration. Use a fixed length string instead by specifying the length with the asterisk.
74	Parameter required	This error occurs when you call a function that requires more parameters than you have supplied.

Err	Message	Description
75	Default arguments not allowed in function definition (use DECLARE)	Default arguments to a function can only appear within the DECLARE statements, not the actual function body definition.
76 - 90	Reserved	Reserved for future use.
91	Function result must be numeric or string	Function results can only be a numeric or string type, not a user defined type.
92	Fixed length string parameter not allowed	Fixed length string parameters in subroutines and functions are not permitted.
93	Function already defined	The function you are trying to define has already been defined.
94	CATCH not allowed here	A CATCH statement is only permitted at the end of a subroutine or function (or at the end of the main program body). A CATCH statement is not permitted after a CATCH ALL statement.
95	Label "label" not declared in block	You have used the GOTO or GOSUB statement with a label that was not declared within the same statement block.
96	Function "function" not defined in program	You have declared a function but have not provided a definition for it.
97-100	Reserved	Reserved for future use.
101	Expression too complex	There are too many nested levels of parentheses in an expression. You should rewrite the expression and break it up into smaller more manageable blocks.
102	Too many arguments	There ate too many arguments in a call to a subroutine or function.
103	RIP command expected	A valid RIP command is expected following the RIP keyword.

Err	Message	Description
104	String expression expected	The RIP command you are trying to execute requires a string parameter at the position indicated by the cursor.
105	Too much global data	Global variables have been defined that exceed the 32K maximum available for data storage.

Run-time errors

Err	Message	Description
0-5	Reserved	Reserved for internal use.
6	ERR_MATH	Caused by a divide by zero, a logarithm of zero or negative number, or square root of a negative number.
7	ERR_ARRAYSUBSCRIPT	Caused by attempting to access an array using an invalid subscript index.
9	ERR_INVALIDFILENUMBER	Caused by using an invalid file number in any file operation.
10	ERR_FILEOPEN	Caused during a file opening operation.
11	ERR_PATH	Caused by an invalid drive or path name when using the MkDir and RmDir commands.
12	ERR_FILERENAME	Caused by an error during a file rename operation using the Name statement.
13	ERR_FILEALREADYOPEN	Caused by using the same file number to open another file.
14	ERR_RUN	Caused when the file name passed to the Run command could not be found.

Editor commands used in the IDE

Key sequence	Description
CTRL C	Page down.
CTRL E	Move cursor up one line.
CTRL G	Delete character.
CTRL H	Backspace and delete.
CTRL I	Next tab stop.
CTRL L	Repeat the last Find/Replace.
CTRL N	Insert new line at the cursor.
CTRL P	Insert a control character.
CTRL R	Page up.
CTRL T	Delete the word to the right.
CTRL V	Toggle insert mode on/off.
CTRL W	Scroll up.
CTRL X	Move the cursor down one line.
CTRL Y	Delete the current line.
CTRL Z	Scroll down.
CTRL J L	Jump to a line number.
CTRL K B	Mark the beginning of a block.
CTRL K C	Copy the block to the new location.
CTRL K H	Hide or un-hide the marked block.
CTRL K K	Mark the end of a block.
CTRL K M	Toggle "Jump-to" markers on/off.
CTRL K N	Save as new file name.
CTRL K Q	Abandon changes to current file.
CTRL K R	Read text from another file into current.

Editor commands used in the IDE

Key sequence	Description
CTRL K S	Save the current file and continue.
CTRL K T	Mark current word.
CTRL K V	Move the marked block.
CTRL K W	Write the marked block to a file.
CTRL K X	Save and close the window.
CTRL K #	Set #(0-3) markers.
CTRL K Y	Delete the marked block.
CTRL O C	Center text between margins.
CTRL O O	Toggle marked text upper/lower case.
CTRL O R	Set right margin for word-wrap.
CTRL O T	The number of blanks to make a Tab.
CTRL O V	Convert marked text to lower case.
CTRL O W	Toggle wordwrap on/off.
CTRL Q A	Open the Replace dialog.
CTRL Q B	Jump to the beginning of marked block.
CTRL Q C	Jump to the end of the file.
CTRL Q D	Move cursor to the end of the line.
CTRL Q E	Jump to the top of the screen.
CTRL Q F	Open the Find dialog.
CTRL Q K	Jump to the block end.
CTRL Q L	Undo current line deletions.
CTRL Q R	Jump to the top of the file.
CTRL Q S	Jump to the start of the line.
CTRL Q X	Jump to the screen bottom.
CTRL Q Y	Delete from cursor to end of line.

RIP commands

For more information on the parameters required by the following RIP commands, refer to the RIPSCRIP.DOC file that is available for download from the MSI BBS on (805) 873-2400

Command	Comment
RIP ARC	Draw circular arc in current color/line thickness
RIP BAR	Draw filled rectangle with fill color/pattern
RIP BEGIN_TEXT	Define a rectangular text region
RIP BEZIER	Draw a bezier curve
RIP BUTTON	Define a Mouse Button
RIP BUTTON_STYLE	Button style definition
RIP CIRCLE	Draw circle in current color and line thickness
RIP COLOR	Set current Drawing Color for graphics
RIP COPY_REGION	Copy screen region up/down
RIP DEFINE	Define a text variable
RIP END_TEXT	End a rectangular text region
RIP ENTER_BLOCK_MODE	Enter block transfer mode with host
RIP ERASE_EOL	Erase current line from cursor to end of line
RIP ERASE_VIEW	Clear Graphics Window to current background color
RIP ERASE_WINDOW	Clears Text Window to current background color
RIP FILE_QUERY	Query existing information on a particular file
RIP FILL	Flood fill screen area with current fill settings
RIP FILL_PATTERN	Set user definable (custom) fill pattern/color
RIP FILL_POLYGON	Draw filled polygon in current color/fill pattern
RIP FILL_STYLE	Set current fill style (predefined) & fill color
RIP FILLED_OVAL	Draw filled ellipse using current color/pattern

Command	Comment
RIP FONT_STYLE	Select current font style, orientation and size
RIP GET_IMAGE	Copy rectangular image to clipboard (as icon)
RIP GOTOXY	Move text cursor to row & column in Text Window
RIP HOME	Move cursor to upper left corner of Text Window
RIP KILL_MOUSE_FIELDS	Destroys all previously defined hot mouse regions
RIP LINE	Draw a line in the current color/line style
RIP LINE_STYLE	Defines a line style and thickness
RIP LOAD_ICON	Loads and displays a disk based icon to screen
RIP MOUSE	Defines a rectangular hot mouse region
RIP MOVE	Move the current drawing position to (X,Y)
RIP NO_MORE	End of RIPscrip Scene
RIP ONE_PALETTE	Set color of 16 color Palette from Master Palette
RIP OVAL	Draw elliptical arc in current color/line style
RIP OVAL_ARC	Draw an elliptical arc
RIP OVAL_PIE_SLICE	Draws an elliptical pie slice
RIP PIE_SLICE	Draws a circular pie slice
RIP PIXEL	Draws a one pixel using current drawing color
RIP POLYGON	Draw polygon in current color/line style
RIP POLYLINE	Draw a Poly Line (multi faceted line)
RIP PUT_IMAGE	Pastes the clipboard contents onto the screen
RIP QUERY	Query the contents of a text variable
RIP READ_SCENE	Playback local .RIP file
RIP RECTANGLE	Draw a rectangle in current color/line style
RIP REGION_TEXT	Display a line of text in rectangular text region
RIP RESET_WINDOWS	Clear Graphics/Text Windows & reset to full screen

Command	Comment
RIP SET_PALETTE	Set 16 color Palette from master 64 color palette
RIP TEXT	Draw text in current font/color at current spot
RIP TEXT_WINDOW	Define the size and location of the Text Window
RIP TEXT_XY	Draw text in current font/color at specific spot
RIP VIEWPORT	Define the size & location of the Graphics Window
RIP WRITE_ICON	Write contents of the clipboard (icon) to disk
RIP WRITE_MODE	Set drawing mode for graphics primitives

The date and time compressed formats used by *wcCODE*

The date and time returned in SearchRec when using FindFirst and FindNext, is passed directly from DOS. The following formulae cover the method used by DOS for storing the time and date as two-byte integers. We include a small program for testing the packing and un-packing of these formats. There is a slight granularity associated with the seconds, as they will always be returned as an even number due to the way they are stored with a division by two.

For encoding and decoding the time passed by DOS, you must first transfer it to a Long as it is possible for the result returned by DOS to be negative. This will produce unpredictable results without the following code being used.

The first sample program does not call the DOS record containing this information as it is merely a demonstration of the compression and decompression process.

The second program uses the FindFirst function and the DOS file record information. Notice that tTime is defined as a long and the value returned by sRec is tested to see whether it is less than zero, if it is then it is made positive by the addition of 65,536.

The time and date encoding is as follows...

- Time = (Hours * 2048) + (Minutes * 32) + (Seconds / 2)
- Date = ((Year - 1980) * 512) + (Month * 32) + Day

The time and date decoding is as follows...

- Hours = Time \ 2048
- Minutes = (Time \ 32) And 63
- Seconds = (Time And 31) * 2
- Days = Date And 31
- Month = (Date \ 32) And 15
- Year = 1980 + (Date \ 512)

First example program for date and time

```
Rem Sample program for converting dates and times
Dim Hour As Integer, Min As Integer, Sec As Integer
Dim Day as Integer, Month As Integer, Year As Integer
Dim tTime As Long : Dim tDate As Long
Hour = 23 : Min = 13 : Sec = 11 : Rem Set dummy time to test
tTime = (Hour * 2048) + (Min * 32) + (Sec / 2)
Day = 31 : Month = 12 : Year = 1994 : Rem Set dummy date to test
tDate = ((Year - 1980) * 512) + (Month * 32) + Day
Rem Now undo them again to see that they work
Hour  = tTime \ 2048
Min = (tTime \ 32) And 63
Sec = (tTime And 31) * 2
Day = tDate And 31
Month = (tDate \ 32) And 15
Year = 1980 + (tDate \ 512)
Print "Time "; hour; ":"; Min; ":"; Sec
Print "Date "; Day; "/"; Month; "/"; Year
Rem End of the main program body
```

Second example program for date and time

```
Dim sRec As SearchRec
Dim Day As Integer, Month As Integer, Year As Integer
Dim Hour As Integer, Min As Integer, Sec As Integer
Dim tTime As Long
If FindFirst("*.*", 0, sRec) = 0 Then
tTime = sRec.Time
If tTime < 0 Then tTime = tTime + 65536
Hour = tTime \ 2048
Min = (tTime \ 32) And 32
Sec = (tTime And 31) * 2
Day = sRec.Date And 31
Month = (sRec.Date \ 32)  And 15
Year = 1980 + (sRec.Date \ 512)
Print sRec.Name, Hour; ":"; Min; ":";
Print Sec, Month; "/"; Day; "/"; Year
End if
```

Boolean bitwise operations

Boolean operators can also be used to manipulate bits according to a logical operation. The following tables indicate the results of bit-wise operations in the *wcCODE* programming language.

The general use of AND, OR and NOT is as relative comparitors, as in **If** 3 > 4 **AND** I < J **Then**...

The **XOr** operator is often used as a bit manipulator in generating passwords that can be later undone with the same algorithm. One value **XOr**ed with another value will return the original value if **XOr**ed again in the same fashion. e.g. 10 **XOr** 12 = 6 and 6 **XOr** 12 = 10.

In each logical operator, the truth table below is applied on a bitwise basis. For example, when **And**ing two values, the first bit in one operand is **And**ed with the first bit in the other operand, and the resulting value is stored in the first bit in the result. When using the predefined values **True** (-1) and **False** (0) all the bits in the values are set one way or the other, so the result is always either True or False.

Truth table for **And**

0	0	0
0	1	0
1	0	0
1	1	1

Truth table for **Eqv**

0	0	1
0	1	0
1	0	0
1	1	1

Truth table for **Imp**

0	0	1
0	1	1
1	0	0
1	1	1

Truth table for **Or**

0	0	0
0	1	1
1	0	1
1	1	1

Truth table for **XOr**

0	0	0
0	1	1
1	0	1
1	1	0

Internal data structures

```
Type ModemProfileRecord
  Version                 As Integer
  ModemName               As String*30
  InitBaud                As Long
  CommPort                As Byte
  CommIRQ                 As Byte
  CarrierDelay            As Byte
  FIFOTrigger             As Byte
  CommBase                As Integer
  RingDelay               As Integer
  DropDTRDelay            As Integer
  PreLogDelay             As Integer
  ResultDelay             As Integer
  ResetDelay              As Integer
  AnswerPhone             As Byte
  DetermineBaud           As Byte
  InterfaceType           As Byte
  RingStr                 As String*60
  AnswerStr               As String*60
  ModemReset              As String*60
  OnHook                  As String*60
  OffHook                 As String*60
  ErrorStr                As String*60
  ModemInit               As String*60
  BaudStrings(1 to 20)    As String*20
  BaudRates(1 to 20)      As Long
  DumpStr                 As String*10
  ResetStr                As String*10
  WriteStr                As String*10
  SetupStr(1 to 3)        As String*40
  Notes(1 to 4)           As String*64
End Type
```

```
Type MakeWildRecord
  Version                  As String*5
  SysopName                As String*25
  FirstCall                As String*25
  Id                       As String*8
  Phone                    As String*25
  BBSName                  As String*30
  FileDatabasePath         As String*67
  UserDatabasePath         As String*67
  NodeInfoPath             As String*67
  MsgAttachPath            As String*67
  DoorFilePath             As String*67
  ModemFilePath            As String*67
  ChatFilePath             As String*67
  LanguagePath             As String*67
  NewUserSec               As String*10
  MonitorType              As Byte
  CloseOption              As Byte
  Network                  As Byte
  DatabaseMode             As Byte
  ScreenBlankMode          As Byte
  ScrollBackBuffer         As Integer
  ExtLTR(1 to 10)          As String*1
  ExtupBatch(1 to 10)      As String*12
  ExtdnBatch(1 to 10)      As String*12
  ExtName(1 to 10)         As String*12
  ExtBatchDriven(1 to 10)  As Byte
  ExtraMemForOverlay       As Byte
  NodeId                   As Byte
  SecTries                 As Byte
```

```
  MaxFileAreas              As Integer
  MaxConfAreas              As Integer
  FirstCallLimit            As Integer
  MWFlags                   As Long
  DateFormat                As String*40
  TimeFormat                As String*40
  RegString                 As String*7
  FlexEventInactivity       As Integer
  FlexEventForceTime        As Time
  LockDriveTable(1 to 26)   As Byte
  DefaultExt                As String*4
  ThumbNailFile             As String*12
  ConsolePassword           As String*14
  ConsoleSec                As Byte
  ChangePhone               As Byte
  ChangeAlias               As Byte
  ChangeBDate               As Byte
  SwapMethod                As Byte
  ModemProfile              As ModemProfileRecord
End Type
```

```
Type MasterRecord
  Version              As String*5
  TotalCalls           As Long
  TotalUsers           As Long
  TotalFiles           As Long
  Totalmessages        As Long
  TempCalls            As Long
  TempMsgs             As Long
  TempDownloads        As Long
  TempUploads          As Long
  TempReset            As DateTime
End Type

Type NodeInfoRecord
  Security             As String*10
  Nodestatus           As Byte
  Userstatus           As Byte
  CallersName          As String*25
  CallerFrom           As String*30
  PrevCaller           As String*50
  BaudRate             As Long
  TimeCalled           As DateTime
  CallerNum            As Long
  TotalCalls           As Long
End Type
```

```
Type ConfRecord
  Name            As String*25
  ConfOp          As String*25
  ShortName       As String*12
  MailType        As Byte
  BullPath        As String*67
  QuesPath        As String*67
  MenuPath        As String*67
  HelpPath        As String*67
  DisplayPath     As String*67
  MsgPath         As String*67
  AttachPath      As String*67
  CFlags          As Integer
  Number          As Integer
  ValidNames      As Byte
  MaxMessages     As Long
End Type
```

```
Type UserRecord
  Name                    As String*25
  From                    As String*30
  Password                As String*14
  UserId                  As Long
  PhoneNumber             As String*15
  DataNumber              As String*15
  FaxNumber               As String*15
  ComputerType            As String*15
  SecLevel                As String*10
  Secondary(1 to 5)       As String*10
  Company                 As String*30
  Address1                As String*30
  Address2                As String*30
  City                    As String*30
  State                   As String*5
  Zip                     As String*10
  Country                 As String*25
  Title                   As String*10
  Alias                   As String*25
  NovellName              As String*8
  Language                As String*8
  Comment(1 to 5)         As String*30
  Sex                     As Byte
  Editor                  As Byte
  MorePrompt              As Byte
  Xpert                   As Byte
  TransferMethod          As Byte
  ScreenDisplay           As Byte
  FileDisplay             As Byte
```

```
  MsgDisplay              As Byte
  LinesPerPage            As Byte
  LastCall                As DateTime
  LastNewFiles            As DateTime
  ExpireDate              As Date
  MemoDate                As Date
  UserSince               As Date
  BirthDate               As Date
  MsgsWritten             As Integer
  Uploads                 As Integer
  Downloads               As Integer
  TimesOn                 As Integer
  TimeLeft                As Integer
  UFlags                  As Long
  DailyDL                 As Long
  DailyDK                 As Long
  TotalUK                 As Long
  TotalDK                 As Long
  MinutesLogged           As Long
  SubscriptionBalance     As Long
  NetmailBalance          As Long
  TCPacket                As Byte
  TCPacker                As String*1
  TCMaxPerConf            As Integer
  TCMaxPerPacket          As Integer
  TCMaxAttAchsize         As Integer
End Type
```

```
Type SecurityProfile
  Name                    As String*10
  ProfileType             As Byte
  ExpiredName             As String*10
  DisplayFile             As String*8
  AccessNode()            As Integer
  AccessDoor()            As Integer
  AccessMenuItem()        As Integer
  AccessProtocol()        As Integer
  SysopStatus             As Byte
  RatioAction             As Byte
  UploadCompensation      As Byte
  MaxRatio                As Byte
  MenuSet                 As String*1
  ExpireDate              As Date
  DailyTimeLimit          As Integer
  MaxLogonTime            As Integer
  VerifyBirthDate         As Integer
  VerifyPhone             As Integer
  MaxDailyDL              As Integer
  MaxDailyDK              As Integer
  MaxKRatio               As Integer
  AccessConfRead()        As Integer
  AccessConfWrite()       As Integer
  AccessConfJoin()        As Integer
  AccessFileList()        As Integer
  AccessFileUp()          As Integer
  AccessFileDown()        As Integer
End Type
```

```
Type MessageHeader
  Number              As Long
  From                As String*70
  FromTitle           As String*10
  FromId              As Long
  To                  As String*70
  ToTitle             As String*10
  ToId                As Long
  Subject             As String*70
  Network             As String*8
  MsgTime             As DateTime
  ReadTime            As DateTime
  Flags               As Integer
  Reference           As Long
  FidoFromZone        As Integer
  FidoFromNet         As Integer
  FidoFromNode        As Integer
  FidoFromPoint       As Integer
  FidoToZone          As Integer
  FidoToNet           As Integer
  FidoToNode          As Integer
  FidoToPoint         As Integer
  InternalAttach      As String*12
  ExternalAttach      As String*12
  Cost                As Long
End Type
```

```
Type FileRecord
  Size                    As Long
  Name                    As String*12
  Password                As String*14
  FileTime                As DateTime
  LastAccess              As DateTime
  Uploader                As String*25
  UploaderId              As Long
  Description             As String*75
  Flags                   As Integer
  Downloads               As Integer
  Cost                    As Integer
  Area                    As Integer
  Keywords(1 to 6)        As String*10
  StoredPath              As String*79
  LongDesc(1 to 15)       As String*79
End Type

Type FileAreaRecord
  Name                    As String*30
  Path                    As String*67
End Type
```

Predefined data variables

```
Dim MakeWild          As MakeWildRecord
Dim MasterInfo        As MasterRecord
Dim NodeInfo          As NodeInfoRecord
Dim Conference        As ConfRecord
Dim User              As UserRecord
Dim UserSec           As SecurityProfile
```

Predefined constants

```
Const True = -1, False = 0
Const On = -1, Off = 0
```

Index

$

$INCLUDE Directive, 229

(

(Line-break) Instruction, 85

+

+ (Concatenation) Operator, 83

@

@Codes, 440
@Codes, connection variables, 443
@Codes, Screen control, 444
@Codes, user record, 441

A

About the IDE, 40
ABS Function, 86
Active window, 32
ACTIVITYLOG Statement, 87
ADDFILE Function, 88
ADDMESSAGE Function, 90
ADDTIME Statement, 92
ADDUSER Function, 93
Allocate variable space, 62
America Online, 16
AND Operator, 67, 95
ANSI Codes, 446
ANSIDETECTED Function, 97
ANSIENABLED Function, 98
Append data to file, 73
APPEND Declaration, 99
Arrays, using and declaring, 58
ASC Function, 100
ASCII, 47
Asterisk, string length, 47
ATLEN Function, 101
ATN Function, 102

B

Backslash, line break, 43
BAUDRATE function, 103
BEEP Statement, 104
Beginners start here, 40
BINARY Declaration, 105
Bitwise operations, 465
Block copy, 35
Block mark text, 34
Block move, 35
Block un-mark, 35
Bold letters, source code, 54
Boolean operations, 465
Boolean truth tables, 465
Break a line, 85
Budding programmer, 40
Bugs in programs, 53
BULLETIN Statement, 106
BULLETINMENU Statement, 107
BYTE Type, 108

C

CALL Statement, 109
CALLERIDRESULT Function, 111
Capital letters in variable names, 56
Caps lock key, IDE, 27
CAPTURE Statement, 112
CARRIER Function, 113
CARRIERCHECK Function, 114
Cascade windows, 38
CASE Statement, 115
CATCH Statement, 117
Caution using wcCODE, 25
CHAIN Statement, 121
CHANGESETTINGS Statement, 123
CHECKBULLETINS Statement, 124
CHECKMAIL Statement, 125
CHECKMAILATLOGON Statement, 126
CHECKNEWSLETTER Statement, 127
CHR Function, 128
Clear primary file, 37
Clear text, 36

Clipboard, 35
Clipboard copy, 35
Clipboard cut, 35
Clipboard paste, 35
Close all windows, 38
Close file function, discussion, 65
CLOSE Statement, 129
Close window, 39
CLREOL Statement, 130
CLS Statement, 131
Code fragments, 82
CODEFRAG.WCC, 82
Colon, statement break, 63
Color codes, 445
COLOR Statement, 132
Comma, print command, 58
COMMENT Statement, 134
Comments in source code, 48
Compile menu option, 37
Compile/Clear primary, 37
Compile/Primary file option, 37
Compile-time errors, 448
Compiling programs, 54
CompuServe, 16
Conditional operators, Boolean, 44
Conection variable @Codes, 443
Conference record variable, 477
Conferences, 17
Conflicting line numbers, 54
ConfRecord, 471
CONST Statement, 135
Constants, 49
Constants, predefined, 477
Contents, help, 39
Continuation, line break character, 43
Convention, lettering styles, 54
Copies, 5, 6, 7
Copy, 5
Copy text, 35
Copy to clipboard, 35
COPYFILE Function, 137
Copyright, 6, 9
COS Function, 138
Cost, 8
CSRLIN Function, 139
CURRENTDATE Statement, 141
CURRENTDATETIME Statement, 142
CURRENTTIME Statement, 144
Customer Support, 15
Cut text, 35
Cut to clipboard, 35

D

Data structures, internal, 467
Data types, user, 50
Data types, user developed, 62
Data variables, predefined, 477
Date/Time decoding, 462
Date/Time encoding, 462
Debugging programs, 61
Decimal, numeric base, 47
Declarations, variables, 45
DECLARE Statement, 145
Declare, functions or subs, 49
Decoding Date/Time, 462
DEFCOLOR Function, 147
DEL Statement, 148
DELAY Statement, 150
Delete text, 36
DELETEMESSAGE Statement, 151
DESQview, 14
device, 5
DIM Statement, 152
Direct assignment, LET, 57
Discussion, not tutorial, 52
DISPLAYFILE Function, 154
DISPLAYGROUPFILES Statement, 155
DISPLAYHELP Statement, 156
DISPLAYPROMPT Statement, 157
DISPLAYSTOPPED Function, 158
DISPLAYTEXTFILE Function, 159
DO Statement, 160
Do...Loop, example, 72
DOOR Statement, 162

DOORMENU Statement, 163
DOSVERSION Function, 164
download, 15
DOWNLOAD Statement, 165

E

Echo conferences, 17
Edit menu option, 34
Edit/Clearremove text, 36
Edit/Copy to clipboard, 35
Edit/Cut to clipboard, 35
Edit/Paste to clipboard, 35
EDITMARKLIST Statement, 166
Editors, 26
ELSE & ELSEIF Statement, 167
ENCODEPASSWORD Function, 169
Encoding Date/Time, 462
END Statement, 171
ENTERMESSAGE Statement, 172
ENVIRON Function, 173
EOF Function, 76, 174
Equivalent keyboard control, IDE, 29
EQV Operator, 175
ERROR Statement, 176
Errors, compile-time, 448
Errors, run-time, 456
Example code, 49
EXISTS Function, 178
EXIT, 64
EXIT Statement, 64, 180
EXP Function, 182
Experienced programmers, 43

F

False, 477
False as a constant, 72
FASTLOGON Function, 183
Fax, 17
Fil/Exit menu option, 34
File/DOS shell menu option, 34
File/Numbers menu option, 34
File/Open menu option, 33
File/Print menu option, 34
File/Save menu option, 33
File/Save-all menu option, 34
File/Save-as menu option, 33
FileAreaRecord, 476
FILEINFO Statement, 184
FileRecord, 476
FILESTATS Statement, 185
Find next, 37
FINDFIRST Function, 186
FINDNEXT Function, 188
FIX Function, 190
FOR Statement, 191
For...Next, loop discussion, 60
FORMATDATE Function, 193
FORMATTIME Function, 195
FREEFILE Function, 197
FUNCTION Statement, 199
Function, discussion, 67

G

GEnie, 16
GET Statement, 202
Get, file record after OPEN, 74
GETCOLOR Function, 204
GETCONFINFO Function, 205
GETFILEAREA Function, 206
GETFILEINFO Function, 207
GETMESSAGE Function, 208
GETMESSAGETEXT Function, 209
GETNEXTFILE Function, 210
GETNEXTMESSAGE Function, 211
GETNEXTUSER Function, 212
GETNODEINFO Statement, 213
GETSECPROFILE Function, 214
GETUSER Function, 215
Global search, 36
Go to line, 37
GOODBYE Statement, 216
GoSub discussion, 54
GOSUB Statement, 217
GOTO Statement, 219

H

HANGUP Statement, 221
HELLOFILES Statement, 222
Help, 15
Help about option, 40
Help/Contents option, 39
Help/Index option, 39
HELPLEVEL Statement, 223
HEX Function, 224
Hexadecimal, numeric base, 47
Hot key, IDE, 26, *27*
HQ BBS, 15

I

IDE, 43
IDE hot key, 26
IDE Hot keys, *27*
IDE keyboard control, *29*
IDE menu commands, 32
IDE, menu bar, 27
IDE, Navigating, *29*
IDE, status line, 27
IDE, using, 26
Identifiers, names and structures, 48
IF Statement, 225
IMP Operator, 227
INCOMM Function, 228
Indenting source code, 61
Index help, 39
Index number, arrays, 58
INDEXADD Statement, 231
INDEXCLOSE Statement, 232
INDEXLOOKUP Function, 233
INDEXNEXTKEY Function, 235
INDEXNEXTREF Function, 237
INDEXOPEN Function, 239
INDEXQUERY Function, 241
INDEXQUERYNEXT Function, 243
Indicators, IDE status line, 27
INKEY Function, 244
Inkey, discussion, 72
Input data from a file, 73
INPUT Declaration, 246
INPUT Statement, 247
INPUTMASK Function, 249
INPUTNUMBER Function, 251
InputNumber, discussion, 63
INPUTYESNO Function, 253
Insert key, IDE, *27*
Installation, 10
Internal data structures, 467
Internal data types, 44
Internet, 16

K

Keyboard commands, 26
Keyword list, 429
Keywords, discussion, 48
keywords, wcCODE, 52

L

Labels instead of line numbers, 54
Language, wcCODE, 44
LCase discussion, 73
Left, String discussion, 69
Len, string length, 69
Length of lines in the IDE, 43
Length of, string, 57
Let, variable assignment, 57
Letters, variable names, 44
License Agreement, 5
Limited Warranty, 8
Line break, 85
Line numbers, wcCODE, 53
List of keywords, 429
List windows, 39
List, @Codes, 440
List, RIP commands, 459
List, Wordstar commands, 457
LOF, 75
LOF Function, 75

M

Machine language, wcCODE, 53
Mailing address, 18

MakeWild record variable, 477
MakeWildRecord, 468
Mandatory string lengths, user types, 62
Marking text, 34
MasterInfo record variable, 477
MasterRecord, 470
Math functions, variables, 55
Maximum length strings, 47
Meaningful variable names, 56
Menu bar, 27
Menu commands, 32
MessageHeader, 475
ModemProfileRecord, 467
Mouse button selection, 31
Move text, 35
MSI HQ BBS, 15
MSI SupportNET, 17

N

Navigating the IDE, 29
Next window, 39
NodeInfo record variable, 477
NodeInfoRecord, 470
Numbers, variables names, 44
Numeric variables, wcCODE, 55
Num-lock key, IDE, 27

O

Off, 477
Off, internal constant, 72
On, 477
On, internal constant, 72
Operators, Boolean, 44
Options, 38
Order entry program, discussion, 61
Output data to a file, 64, 73

P

Parentheses, grouping for maths, 44
Paste text, 35
Paste to clipboard, 35
Period with variables, 62
Permitted uses, 10
Pigeon holes, 55
Powerful language, 52
Predefined constants, 477
Predefined data types, 44
Predefined data variables, 477
Previous window, 39
Primary file, 37
Programming editors, 26
Programming interface, 26
Put, data to file discussion, 74

Q

Quickest calculation times, 55

R

Random, file open discussion, 73
Record structures, discussion, 61
Record, ConfRecord, 471
Record, FileAreaRecord, 476
Record, FileRecord, 476
Record, MakeWildRecord, 468
Record, MasterRecord, 470
Record, MessageHeader, 475
Record, ModemProfileRecord, 467
Record, NodeInfoRecord, 470
Record, SecurityProfile, 474
Record, UserRecord, 472
Records, internal data structures, 467
Refund, 5
Registration, 16
REM, 70
REM Statement, 48, 70
Repetitive loop instructions, 60
Replace text, 36
Reserved keywords, 429
Reserved words, wcCODE discussion, 68
Resize window, 38
return, 9
RIP commands, list, 459
Run-time errors, 456

S

Sample programs, 51

Satisfaction Guarantee, 9
Screen control @Codes, 444
Search/Direction option, 36
Search/Find again, 37
Search/Find menu option, 36
Search/Global option, 36
Search/Go to line, 37
Search/Range option, 36
Search/Replace option, 36
SecurityProfile, 474
Seek,, 75
Semicolon, print display discussion, 60
Set options, 38
Size window, 38
Software License Agreement, 5
Source code from manual, 82
Special Symbols, 44
Status line, IDE, 27
Storage space for variables, wcCODE, 57
String length, 57
Strings, use in wcCODE, 47
Structure of user types, 50
Subroutine discussion, 49, 67
Subscript for array variables, 58
Symbols, 44
Syntax of wcCODE, 59
Sysop, 16
System information @Codes, 440

T

Technical Support, 14, 15, 17
Temporary clipboard, 35
Tile windows, 38
Time/Date decoding, 462
Time/Date encoding, 462
Trim, String function discussion, 69
True, 477
True, internal constant, 72
Truth tables, 465
Tutorial, really a discussion, 52
Two dimensional variables, 58
Type ConfRecord, 471
Type FileAreaRecord, 476
Type FileRecord, 476
Type MakeWildRecord, 468
Type MasterRecord, 470
Type MessageHeader, 475
Type ModemProfileRecord, 467
Type NodeInfoRecord, 470
Type SecurityProfile, 474
Type UserRecord, 472
Types, internal data structures, 44

U

UCASE Function, 73
UCase, function discussion, 72
Underline character in variable names, 57
Un-mark text, 35
Updates, 18
User data types, 50
User record, 44
User record @Codes, 441
User record variable, 477
UserRecord, 472
UserSec record variable, 477

V

Variable names, 44
Variable parameters, Sub/Function, 56
Variables, 55
Variables, predefined, 477

W

Warning using wcCODE, 25
Warranty, 8
wcCODE language, 44
Wend, statement discussion, 76
While, statement discussion, 76
Wildcat! BBS, 15
Window/Cascade option, 38
Window/Close, 65
Window/Close all option, 38
Window/Close option, 39
Window/List option, 39
Window/Next, 60

Window/Next option, 39
Window/Previous, 39
Window/Size option, 38
Window/Tile option, 38
Window/Zoom option, 38

Wordstar commands, 26, 457

Z

Zoom windows, 38

7 29501 10406 1

MSI

MUSTANG
SOFTWARE
INC.